Rui Abreu de Carvalho Ferreira

Privacidade e Selecção de Identidades

Privacy and Identity Selection

Rui Abreu de Carvalho Ferreira

Privacidade e Selecção de Identidades

Privacy and Identity Selection

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Dr. Rui Aguiar, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e da Dra. Susana Sargento, Professora Auxiliar Convidada do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

o júri / the jury

presidente / president

Doutor Atílio Manuel da Silva Gameiro
Professor Associado da Universidade de Aveiro

vogais / examiners committee

Doutor Edmundo Heitor Silva Monteiro
Professor Associado com Agregação do Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra

Doutor Rui Luís Andrade Aguiar
Professor Auxiliar da Universidade de Aveiro (Orientador)

Doutora Susana Isabel Barreto de Miranda Sargento
Professora Auxiliar Convidada da Universidade de Aveiro(Co-Orientadora)

agradecimentos /
acknowledgements

Gostaria de expressar o meu agradecimento ao Prof. Doutor Rui Aguiar e á Prof. Dra. Susana Sargento pela apoio científico e pedagógico dado no decurso da elaboração deste trabalho.

Agradeço a todos os colegas do Instituto de Telecomunicações - Aveiro, em especial ao Engenheiro Alfredo Matos pela preciosa ajuda dada na realização deste trabalho

Palavras Chave Privacidade, Identidade

Resumo Este trabalho aborda algumas das tecnologias emergentes, que visam providenciar modelos com suporte para identidade, sobre a arquitectura da internet. Enquanto que tais propostas providenciam de facto modelos de identidade mais adequados para a Web 2.0, ignoram que a arquitectura sobre a qual assentam não pode suportar os requisitos de privacidade necessários. A fim de demonstrar que é necessário providenciar garantias de privacidade nas camadas inferiores da arquitectura existentes, este trabalho apresenta uma implementação de demonstração que explora vulnerabilidades da arquitectura existente a ataques de privacidade, assim como uma implementação que dá garantias de privacidade para múltiplas identidades.

Keywords Privacy, Identity, Unlinkability

Abstract This work presents some of the technologies, that aim at introducing user centric identity models over the existing Internet architecture. While such technologies do provide suitable identity models for the Web 2.0, they neglect that the underlying Internet architecture does not support the necessary privacy requirements for such models. As a demonstration that the underlying network layers must provide some privacy guarantees, this work presents an implementation that explores privacy vulnerabilities and proceeds to propose mechanisms that ensure privacy between multiple identities.

Contents

List of Figures

List of Tables

Acronyms

AN	Access Network.
DNS	Domain Name System.
ICANN	Internet Corporation for Assigned Names and Numbers.
IdP	Identity Provider.
IP	Internet Protocol.
NGN	Next Generation Networks.
OP	OpenID Provider.
PKI	Public Key Infrastructure.
RP	Relying Party.
SIP	Session Initiation Protocol.
SLO	Single Logout.
SP	Service Provider.
SSO	Single Sign On.
TLS	Transport Layer Security.
UA	User Agent.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.

VID	Virtual Identity.
VIF	Virtual Interface.
VIP	Virtual Identity Proxy.
XRI	Extensible Resource Identifier.

Chapter 1

Introduction

Nowadays it is not uncommon for users to travel across networks, while remaining always reachable. During his movements across different networks, the user will interact with a large variety of services. It could be said that the user assumes multiple identities, each identity interacting with several services, with different purposes in mind. For example it is not uncommon to consider that an individual's work affairs and personal affairs, are separated aspects of one's life. This notion of separation, in real life as in electronic context, is normally considered for privacy reasons. The actions the user performs under the mantle of each of his virtual identities, are usually unrelated, and it is in the user's best interest they remain so. Privacy concerns arise when actions performed under different identities are easily linkable. Furthermore, privacy is not only a social concern it is also a legal requirement in many countries, where existing privacy legislation addresses the protection of intrusion to individual's privacy.

This kind of privacy issues are not new, however most users lack the necessary awareness and will find themselves increasingly vulnerable, as all IP Next Generation Networks emerge, making the user increasingly exposed to these privacy threats. The need for permanent user availability and phenomenons like "Social Networking" are redrawing the Internet as an Identity based infrastructure. Users and services exchange information based on a diffuse notion of identity that can take various forms, from the traditional email address or username to the Uniform Resource Identifier of a blog, photolog or vlog. However the user's identity is scattered across a multitude of services and becomes difficult to perceive, making it hard for the user to take control of his identity.

To address these issues several identity concepts have emerged, that promote identity models with stronger authentication and improved user privacy. Emerging identity frameworks like OpenID or SAML , envision identity management systems as an additional layer on top of the OSI stack(fig. 1.1). In practice such solutions allow the user to select which virtual identity they wish to use, when contacting different services. Ideally different user identities would be unlikable, in order to preserve user privacy. While they have succeed in providing users with increased control over their identity, their work is confined on top of the application layer, and neglects that the underlying layers do not provide the necessary privacy requirements for identity privacy. User identities can still be linked through the identifiers used in the lower layers.

The work in [37, 27], already outlines the requirements for unlinkable identities within the lower layers of the OSI stack. Proposing that in order to have unlinkable identities, a cross layer

approach is required, that guarantees different identifiers for each identity. This work attempts to demonstrate that user identities are in fact linkable through L2, L3 and L7 identifiers, regardless of identity mechanisms on top of the application layer and proposes an implementation that imbues a browser implementation, with the capability to provide unlinkable identities to the user.

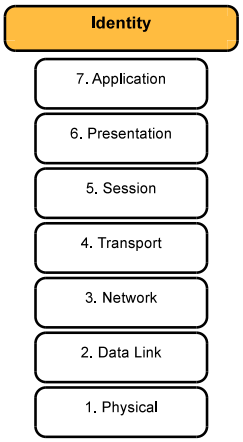

Figure 1.1: A modified OSI stack with an additional Identity Layer

1.1 Objectives

This work aims to:

1. Study privacy concepts
2. Study some of the existing Identity Frameworks
3. Identify what kind of attacks can be used for user identity correlation
4. Apply state of the art proposals[27] to solve identity correlation attacks
5. Solve existing shortcomings in terms of identity selection

The first and second topics involve the study of relevant privacy concepts and of the existing frameworks within the context of identity privacy. The third, forth and fifth topics will involve the creation of suitable implementations that not only demonstrate how an attacker could explore the lack of adequate identity privacy protection, but also achieves a successfull mitigation solution.

1.2 Disposition

This document is organized in 6 chapters, the first one being the present introduction and the sixth one a summary of the conclusions taken from this work. The remaining chapters are organized in the following manner:

- Chapter 2 is an overview of emerging Identity Management Frameworks based on the concept of federated identity. It presents the federated identity model and outlines how new identity frameworks are converging to provide "user-centric" identity.
- Chapter 3 presents the attacker model considered throughout this work, some techniques an attacker may use as well as implementations of some of the described attacks.
- Chapter 4 presents a working implementation to guarantee unlikable user identities.
- Chapter 5 presents the scenarios considered for implementation testing and the obtained results.

Chapter 2

Digital Identity Frameworks

2.1 Identity Privacy

The term Privacy refers to seclusion from the view of others. Privacy is said to exist if the usage, release and dissemination of personal information can be controlled. An invasion of privacy can be seen as, a situation where an individual cannot control the exposure and usage of his personal information.

2.1.1 Concepts

2.1.1.1 Anonymity

In order to enable anonymity for an individual, a set of other individuals must exist with the same attributes. Essentially making an individual indistinguishable from a set of others(the "anonymity set"[43]). The larger the anonymity set is, the harder will be for an attacker to distinguish an individual from the remaining individuals in the set. Ideally the anonymity set will be such, that the attacker will not be able to sufficiently single out an individual.

2.1.1.2 Linkability

Linkability is the ability to determine if two items are related or not. In terms of user identities, linkability is the ability to determine if two identities are related to the same individual. In the context of identity management, two identities are linkable if they share some sort of attribute, that allows one to conclude that both identities belong to the same individual. For two identities to be unlinkable they must not share any kind of identifier or attribute, that would allow that sort of conclusion to be inferred.

2.1.1.3 Observability

Observability is the possibility that an attacker will gain information by observing message exchange. The attacker can be either a party involved in the message exchange or an eavesdropper.

While a party involved in a message exchanged, can not be prevented from observing the exchange, in an ideal situation an external eavesdropper wouldn't even be able to determine that the message exchange occurred.

2.1.1.4 Pseudonymity

Pseudonymity refers to the use of pseudonyms as identifiers. A pseudonym is an identifier that will have no direct relation to the user's real identity, that will be used to refer to the user's identity in one or more contexts. Pseudonyms are usually employed to avoid having to use the user's name, or other attribute directly related to the user.

The pseudonym can be any kind of identifier. Depending on the kind of identifier a pseudonym may inherit additional properties. For example if a user employs a web site URI as a pseudonym, the pseudonym will be easily linkable to the information in the site. This kind linkability between the identifier and additional identity information is sometimes desirable for usability purposes, for example to provide an easy and recognizable way to refer to a specific identity. Throughout this work most the discussed identifiers will in fact be pseudonyms, being commonly referred as "identifiers".

2.1.1.5 Identity

The term Identity usually refers to a representation of an entity that consists of one or more attributes related to the identified entity[20, 31]. Such representation must characterized the described entity in a way that differentiates the entity from all other entities in the context where the Identity is employed.

Attributes within an Identity describe characteristics of the identified entity. If an attribute is unique for an identity within a given context then within that context such attribute can be seen as an identifier, that is an attribute that can be used to refer to the identity.

Since each user identity is characterized by one or more identifiers, valid within one or more contexts. For two identities to be unlinkable in any context, they must not share any of these identifiers. The following considerations should be made when addressing identifier usage, for user identities:

- How easy it is to get hold of a user's identity identifier
- How easy it is to dereference an identifier to the user's identity
- How much information can be gathered by dereferencing an identifier

2.1.2 Privacy Information Practices

Various principle sets for information practices exist. Varying for each country's legal background[26, 21], the following subset seems to be rather common regarding collection of user information:

- Collection purposes: Prior to collecting any user information, the user must be informed of the purpose of the collection
- User Consent: User information collection shall not take place, without the user's consent
- Use and Disclosure: Collected user information, shall not be used or disclosed for purposes other than those for which it was collected
- Information Access: Upon request a user will have access to all of his personal information stored by a third party and shall be able to amend it as appropriate

2.1.3 Privacy Policy

Privacy policies are a way for organizations to express privacy practices, employed for one or more provided services. Privacy policies are the first step, torwards informed user consent for data collection. Such statements should include:

- What user information will be collected
- The purpose given to the collected information
- The recipients of the collected information
- User's rights to access and modify the collected information

User's tend to see presentation of privacy policies by Service Providers(SP), as a sign of good faith. However, there is no easy solution to verify that Service Providers actually adhere to the stated privacy policies. Privacy policy statements are quite common in web sites and most Service Providers state that no information will be collected without user consent. However they are unclear regarding what will be the intended used for the collected information, specially when it comes to information disclosure to third parties. Even when claiming that user information will not be disclosed without consent, they contradict themselves by stating that information will always be shared with their business partners[42].

This kind of loopholes in privacy policies seems to suggest that, the user cannot actually control collection and dissemination of information, even if the Service Provider complies with his privacy policy. Furthermore web applications work using a client-server model, where the user initiates the interactions, this will make some his network identifiers(L3 and perhaps L7) available to the Service Provider with or without user consent.

2.2 Federated Identity Model

Up until recently the majority of Service Providers implemented an identity model centered around the Service Provider. As a result the user was required to create some sort of account in the service provider, typically containing a username, a password and an email address. While this works well for a single service it quickly becomes unmanageable if the user interacts with a large number of Service Providers: the user will have to remember several passwords as well as different usernames, since username collision among users is a common occurrence. Besides having to recall authentication information for all the Service Providers, the user will probably provide some additional contact information to the service provider, like an email address or a phone number, and this information will be stored in all the different Service Providers. If the user's contact information changes he will have to update his contact in all of Service Providers.

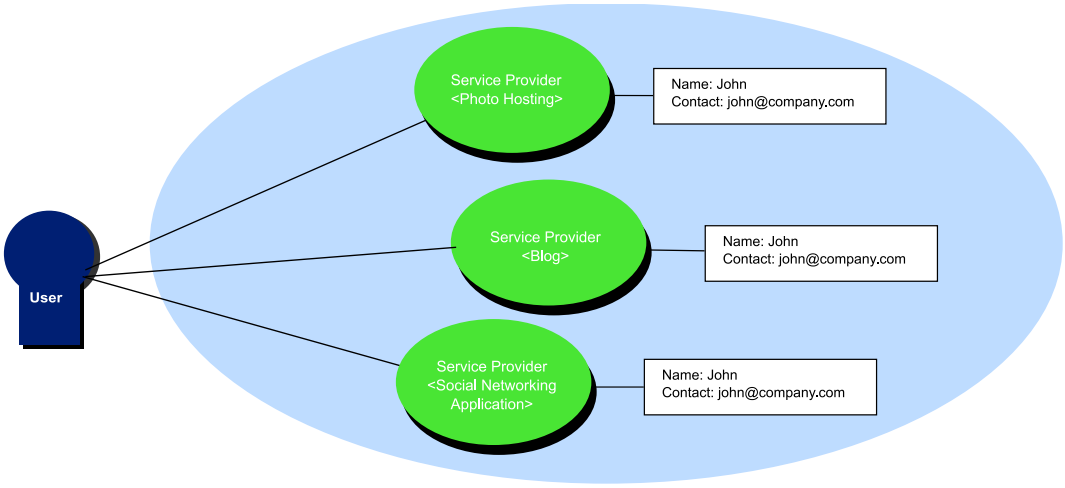

Figure 2.1: SP centered Identity model

This model has significant redundancy and scalability problems, as it scatters the user information across the different Service Providers(fig. 2.1), making it hard for the user to manage his information. Additionally users have expressed that privacy concerns over the dissemination of their private information[1], are the main hindrance preventing them from adopting online commerce solutions[19], which prompted the Service Providers to look into models that address user privacy concerns.

The typical user sees his identity as a conjunction of different attributes even if these are stored by different Service Providers, however the perception that other users have of his identity might be completely different since they might not be able to see all his attributes. As an example in a scenario(fig. 2.2) where a user interacts with three different Service Providers, other users might have a different perception of that user's identity based on what service providers they use themselves. Other users may perceive one or more different identities when in fact only one exists depending on whether they use one, two or the three Service Providers, which shows that the identity the user wishes to project may differ considerably from the identity perceived by other

users.

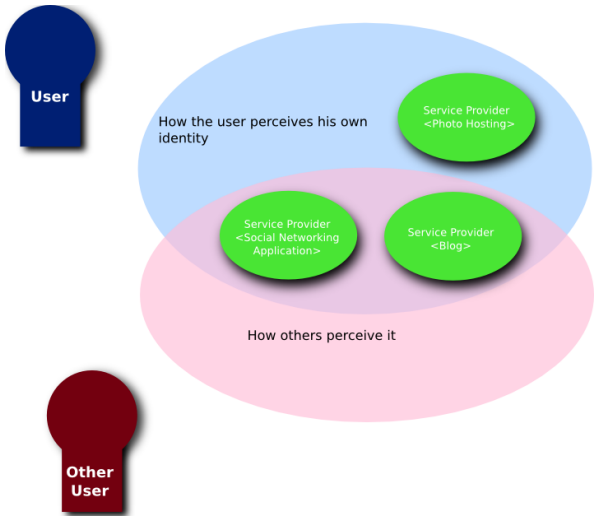

Figure 2.2: The different perception different users get of what should be the same identity

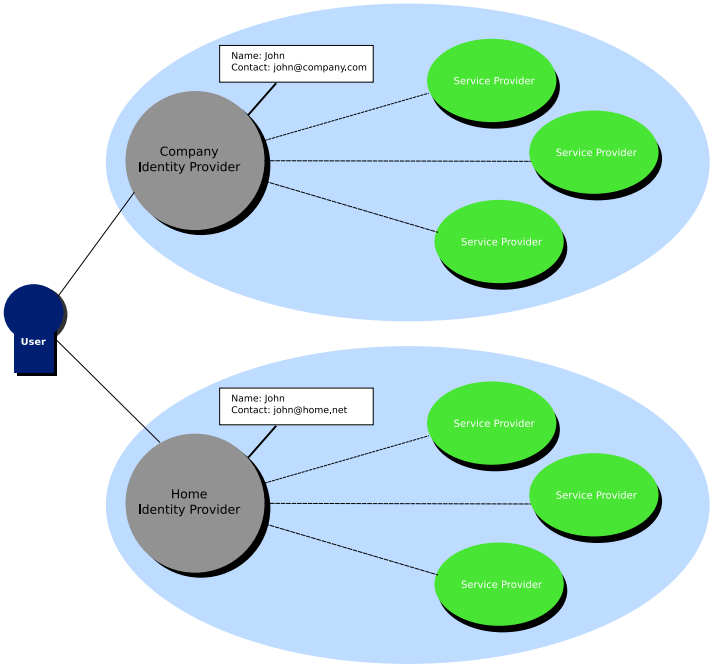

Figure 2.3: "User-centric" Identity model

In order to address these issues the user's Identity must be detached from the service provider, this is possible through the introduction of an additional entity called an Identity Provider(IdP). The IdP is essentially a special Service Provider responsible for the management of the user's identity. Instead of storing the user's profile at each of the Service Providers the user information is now stored at the Identity Provider. This model is called "user-centric"(fig. 2.3) in the sense that it moves the user's identity away from the Service Provider and closer to the user, allowing the

users to have increased control over the dissemination of their identifiers and private information across the different services they use.

This model is quite similar to the one seen in offline identity transactions, where the user simply travels around with a document containing assertions certified by a third party. An example of this could be a driver's license: the license is issued by a third party and the user presents it to a Service Provider when he deems it necessary. The digital model possesses the added advantage that the user may choose to only present a subset from the user's information. If the user wishes to access a service that does not require his full name, then the Identity Provider won't supply it to the service provider thus reducing user information disclosure.

The fact that the Service Provider will accept an assertion by the Identity Provider doesn't mean that some sort trust relation exists between the Service Provider and the Identity Provider. The kind of trust required between both entities depends highly on the type of assertions being exchanged, as the context in which they are being exchanged. For a Service Provider inside a company it would probably make sense to only accept assertions from a Identity Provider inside the company, while for Internet Business Provider it would make sense to accept assertions from all Identity Providers in order to maximize the number of clients.

2.2.1 Entities

2.2.1.1 Service Provider

A Service Provider is an entity that provides services to other entities. Usually providing subscription based service to individuals.

2.2.1.2 Identity Provider

The Identity Provider(IdP) is the entity responsible for the management of the user identity, it is an entity to whom the user entrusts the ability to make assertions about his identity. If a service provider requires identity related information about a user, it will have to contact the IdP in order to get that information. Since the Identity Provider will be making assertions regarding the user's identity, a minimal amount of trust between the user and the IdP is assumed to exist.

2.2.2 Mechanisms

2.2.2.1 Identity Federation

Identity Federation is a process in which two parties agree on a set identifiers used to refer to an Identity. When a user first makes contact with a Service Provider, the user's Identity will be federated, the user's Identity Provider and the Service Provider will agree on a set of identifiers used among them to refer to the user's identity, these identifiers may be:

- globally unique, and thus context independent
- valid only in the context of a specific Service Provider and a specific IdP

Globally unique identifiers allow for a user identity to be correlated across Service Providers, these identifiers are usually employed when a user wishes to be identifiable by other users across several Service Providers. Context dependent identifiers are valid only between a specific SP and IdP, ensuring that even if multiple SPs were to exchange user information, user identifiers would not match.

Depending on the Service Provider the Identity Federation process can be a seamless process for the user, or more similar to traditional registration process. The Service Provider might require that the Identity Provider provide additional attributes about a user before federation can occur like a full name or an e-mail contact for example, requiring the user to intervene in the Federation process.

2.2.2.2 Single Sign On

The term Single Sign On(SSO) is used to refer to an authentication method, that enables a user to only present his authentication credentials once while gaining access to several different services. This was common among Service Providers within the same domain, for example inside a company an employee would have to authenticate only once and would immediately be authenticated to all the necessary company services.

This is the traditional concept of SSO, however as proposals for SSO across different domains appeared with SAML[30] and OpenID[41], the meaning of SSO became ambiguous. While they also refer to SSO as a mechanism where the user only presents his credentials once, the user might not be immediately authenticated at the Service Providers. For the user to be recognized by a Service Provider he must first select which identity to present to the Service Provider.

2.2.3 Benefits and Pitfalls

This model entails significant benefits for all parties, the User enjoys the benefits of Single Sig On having to log in only once using one set of credentials, without revealing the credentials to any of the Service Providers. Service Providers can delegate user account management tasks, such as password resets or profile information changes, while being able to acquire accurate updated user information. The Identity Provider can focus on the improvement of user authentication methods, being able to provide services most regular Service Providers never would(phishin resistant authentication methods for example).

Not unlike most outsourcing, the Federated Identity model is not without it's risks. Since this model introduces the notion of a cross-domain identity, user information will be exchanged across different domains, in an ideal scenario all parties would secure their communication channels against eavesdropping based attacks, but such is not always the case. Also the cost of a stolen

user credential is now much higher, if an attacker can steal a user credential, he will gain access to all Service Providers that the user federated with.

This makes the Identity Provider a probable target for attackers seeking to compromise user privacy. Also, if the Identity Provider is ill-intended, it might disclosure user information against the user's will. This places the need a large amount of trust between the User and the Identity Provider, the user is after all trusting the IdP with his authentication credentials as well as with access to all the Service Providers the user has federated with.

2.3 OpenID

OpenID is a community driven effort and was originally developed as a lightweight, decentralized way to authenticate commenters in order to prevent spam in LiveJournal's blogs. It was designed from the beginning to be used on HTTP based applications and to be as simple as possible requiring no additional support on the user's terminal. It rapidly gained adepts among the "social networking" community as method of authentication across different blogs.

2.3.1 OpenID User Identifiers

In OpenID a globally unique Identifier exists for each user identity. This Identifier can be used to reference an identity in any context, and must be resolvable into an OpenID authentication service. The OpenID 2.0 specification[41] states that two types of identifiers are acceptable, Uniform Resource Identifiers[14] and Extendable Resource Identifiers[46](XRI).

URIs were initially chosen as identifiers in OpenID because they were already an widely adopted identifier that was not only a globally unique identifier but also an easily recognizable one. Since "Uniform Resource Identifiers (URI) provide a simple and extensible means for identifying a resource"[14], they are eligible as a suitable identifier for user identities. With the added benefit that they allow a user to easily tie content to his identity, for example using the URI of his personal page as an identifier, which is particularly convenient in blogging environments were a user's identifier would be the URI of his own blog.

XRIs are an OASIS specification that extends URIs with an additional resolution mechanism. While URIs rely on DNS for the resolution process managed by ICANN, XRI's resolution process on the other hand is managed by XRI.org. Since XRIs are based on the URI specification and thus retain it's main characteristics, they are suitable as identifiers for OpenID and have been introduced as OpenID identifiers in the OpenID 2.0 specification.

```
http://www.johnsmith.com
xri://=john.smith
```

Figure 2.4: Two suitable OpenID identifiers. The first is an URI and the second an XRI

The use of URI based unique identifiers will immediately expose user information to any third party that comes across the identifier. Since user identifiers are URIs, these identifiers can be easily dereferenced and lead to the user's Identity Provider. OpenID was intentionally designed in this manner. It can be said it assumes an "open" trust model, where easy dereferencing of user information is desirable.

2.3.2 Authentication

Authentication is a process through which a user can prove that he is the owner of an identifier by presenting credentials associated with that identifier. In OpenID for a user to authenticate with a Service Provider he must prove that he owns a specific OpenID identifier. For the service provider to be able to determine if the user's identify claim is true, it must contact the Identity Provider(IdP) that manages the claimed identifier(fig. 2.5).

For a Service Provider to authenticate a user it must:

1. Discover which IdP manages the user's claimed identifier
2. Have the IdP determine if the user's claim is valid

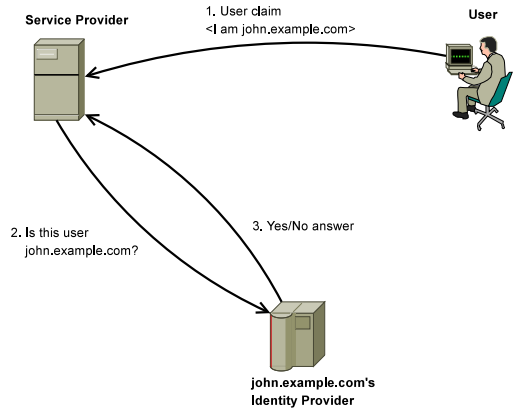

Figure 2.5: A user claiming to be john.example.com

2.3.2.1 Identity Provider Discovery

Identity Provider Discovery is a resolution process that dereferences the identifier supplied by the user to an OpenID Identity Provider, that can be contacted by the Service Provider to determine if the user's identity claim is valid.

In the OpenID 1.1 specification the discovery process consists of fetching the HTML document at the claimed URI identifier, the fetched document's HEAD section is then parsed for an *openid.server* declaration(fig. 2.6).

```
<head>
<link rel="openid.server" href="http://john.idprovider.com/">
</head>
```

Figure 2.6: An OpenID server declaration inside an html HEAD section

With the introduction of OpenID 2.0[41] two additional discovery mechanisms were introduced:

1. XRI Resolution
2. Yadis protocol

Since XRI identifiers were introduced in OpenID 2.0 the discovery process must now be able to resolve XRI addresses according to the XRI resolution specification[46]. The Yadis protocol[38] is a protocol for resolution of URIs into sets of services, in the context of OpenID Yadis is used to determine the address of an Identity Provider from an URI. Both Yadis and XRI Resolution result in a XRDS document containing at least one OpenID Identity Provider service(fig. 2.7), if Yadis fails when resolving an URI then the OpenID 1.1 discovery method is used.

```
<Service xmlns="xri://$xrd*($v*2.0)">
  <Type>http://specs.openid.net/auth/2.0/signon</Type>
  <URI>http://john.idprovider.com/openid2</URI>
</Service>
```

Figure 2.7: An OpenID 2.0 server declaration inside an XRDS document

Upon successfull execution of the discovery process a Service Provider will have the URI of the IdP as well as the versions of the OpenID specifications the IdP supports.

2.3.2.2 OpenID Authentication Workflow

Figure 2.8 shows a typical openid authentication workflow. Upon visiting a service provider that requires authentication, the End-User presents his OpenID Identifier, the End-User's browser is then redirected from the Service Provider to the Identity Provider. If necessary, the End-User will then authenticate against the Identity Provider typically using a username and password, if the Identity Provider considers that the End-User is authenticated it will then redirect the End-Users' browser back to the Service Provider along with a cryptographically signed message that confirms the user's identity. For the Service Provider to be able to verify the signed message, it must establish an association with the Identity Provider in order to negotiate the necessary cryptographic key. Such a key is usually negotiated using Diffie-Helman and generated between the Service Provider and the Identity Provider, when necessary. Such keys refer to an association between the SP and the IdP, and will most likely be reused by the Service Provider to verify credentials from different user using the same Identity Provider. Since these keys are generated

on the fly, no previous trust infrastructure is in place, or at least the OpenID specification does not address such scenarios.

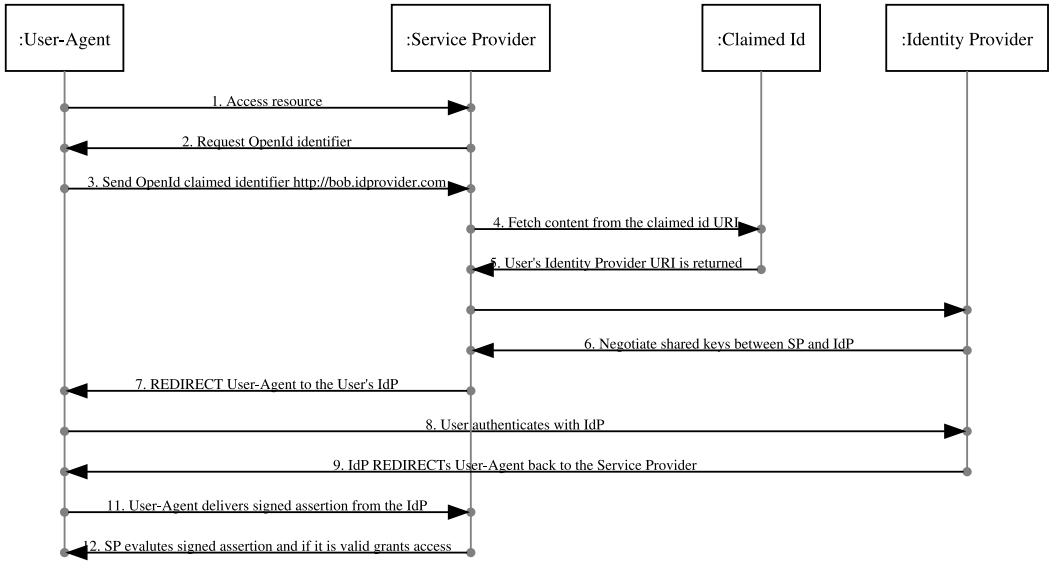

Figure 2.8: Typical workflow for the OpenID authentication

This workflow allows the end-user's browser to enter in a OpenID authentication process without any kind of additional OpenId support in the browser. However this is also the workflow that exposes the top security concern in OpenID, the assumption that the Service Provider will in fact redirect the end-user's browser to his identity provider. A malicious service provider might redirect the end-user's browser to a malicious identity provider, that would employ a phishing attack to capture the user's authentication credentials.

A typically workaround for this would be to employ a fishing resistant authentication mechanism, or introduction of browser support for OpenID that would not require redirection of the end-user's browser from the service provider There is already one upcoming draft specification[29] for integration of OpenId with Cardspace authentication that enables such support in the browser.

2.3.3 Identity Delegation

One of the main concerns that quickly emerges regarding identity frameworks like OpenID is the matter of trust. Finding a trustworthy Identity Provider to whom a user will entrust the management of his digital identity is not an easy task. Since OpenID is completely distributed a user could run his own Identity Provider, but most users can't or don't desire to do so because it requires that the user has a minimal level of know-how as well as some support infrastructure to host the IdP.

One way to achieve some independence from the identity provider in use is through the use of delegation. Suppose a user wishes to use his personal web site's URI http://www.john.com/

has an OpenID Identifier and he already possesses an OpenID account at an existing Identity Provider http://john.idprovider.com. The user could delegate(fig 2.9) the authentication of the http://www.john.com URI to the identity provider at http://john.idprovider.com.

This way if the Identity Provider at http://idprovider.com were to disappear or if the user deemed that this identity provider was no longer trustworthy he could simply switch to a different identity provider(fig 2.10). As long as the user controls his Identifier he can switch seamlessly between Identity Providers.

```
<link rel="openid.server"
href="http://idprovider.com/openid/server/">
<link rel="openid.delegate" href="http://john.idprovider.com/">
```

Figure 2.9: Delegation of an identity to another identity http://john.idprovider.com

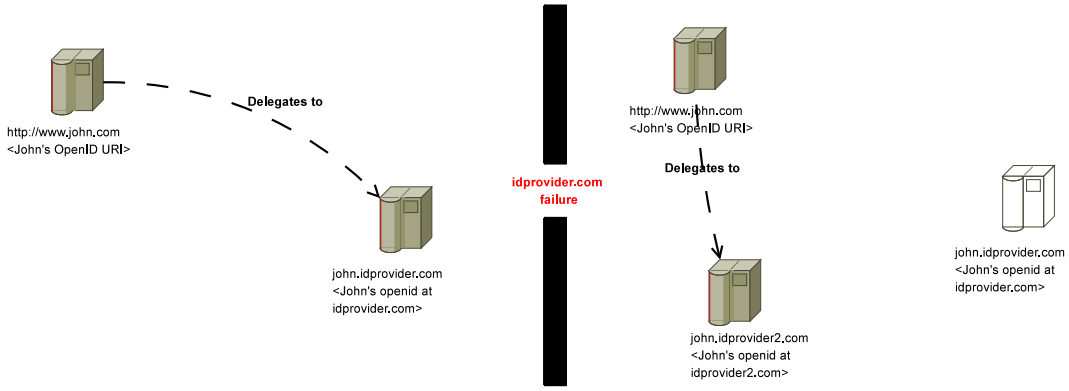

Figure 2.10: The user can seamlessly switch identity providers

2.3.4 Directed Identity

One of the prime concerns regarding the use of a globally unique identifier is leakage of the identifier between service providers. If a user exposes his identifier to two service providers and these service providers exchange these identifiers and the associated information without the user's consent, the involved providers may be able to infer additional information about the user. While in some cases this information exchange might be beneficial to the user, the premises behind the "user-centric" concept is that the user has control over the dissemination of his information. This privacy shortage comes from the fact that OpenID 1.1[45] only supports globally unique identifiers.

In order to address this the OpenID 2.0[41] specification introduces the concept of "Directed Identity". In the workflow presented earlier(fig. 2.8), the user presented his OpenID Identifier to the service provider upon an authentication request. As of OpenID 2.0 a user may instead present an Identity Provider Identifier(the URI of the Identity provider), this identifier resolves directly to the Identity Provider and thus the Service Provider will only be aware of the user's OpenID

Identifier in the end of authentication process.

Upon being redirected from the Service Provider to the Identity Provider, the user must then negotiate with the Identity Provider which OpenID Identifier will be used, the user can choose to use a regular OpenID URI or a pseudonym Identifier. The user will then be redirected back to the service provider, with a signed message asserting which identifier is to be used. If the user uses different identifiers for each Service Provider, then the service providers will not be able to correlate the user's information based on his OpenID Identifier.

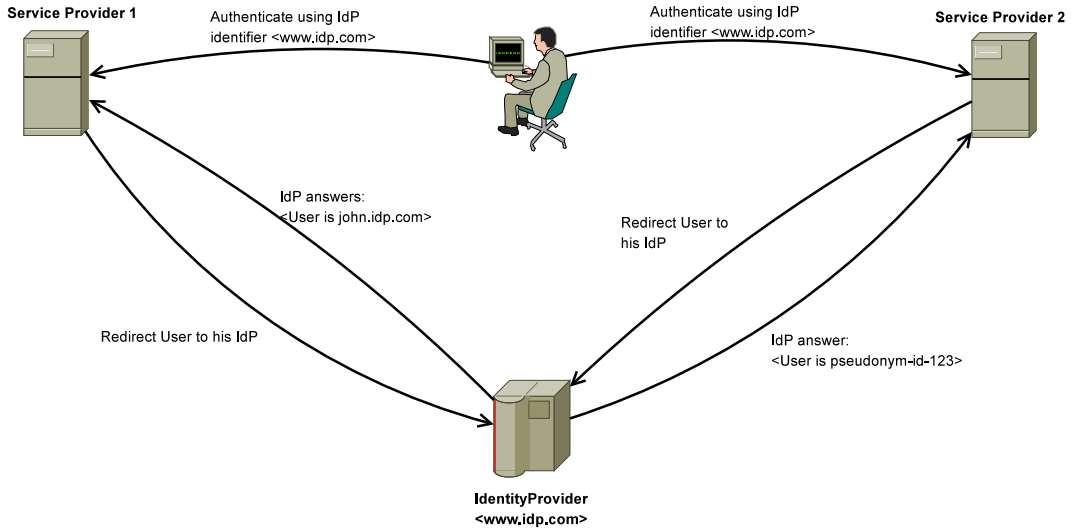

Figure 2.11: The same user uses different OpenID identifiers to different service providers while using the same IdP

2.3.5 Privacy and Security Issues

Being that in OpenID user information is most likely tied to the OpenID identifier, the use of such a public identifier as an URI will most likely lead to leakage of user information[50]. This contrasts with the basic concept of "user-centric" identity that a user should have absolute control over the dissemination of his identity information. OpenID actually encourages disclosure of identity information in this manner, mainly due to its uprising in the context of social networking applications, "as an architecture biased torward broadly sharing user information"[36]. In a scenario where a user wishes zero information disclosure, OpenID is probably not the appropriate solution.

Also since OpenID lies on top of the existing web infrastructure. It advises use of TLS when it considers necessary and depends on DNS for entity resolution but it does not address security and privacy concerns of the underlying layers. If HTTP were to be used for exchange of identity claims, there would be no guaranties regarding confidentiality, similarly if the underlying DNS infrastructure was to be compromised then the notion of ownership of an OpenID identifier would also be compromised.

2.4 SAML 2.0

The Security Assertion Markup Language is an OASIS specification that defines a XML encoded format for exchange of Authentication, Authorization and Attribute information between entities. The SAML 2.0 specification represents the convergence of SAML 1.1, Liberty Alliance's ID-FF Framework[49] and Shibboleth 1.3[16] specifications. SAML aims at being as extensible as possible by dividing its specification in three main parts:

- SAML Core
- SAML Profiles
- SAML Bindings

This separation aims to make SAML more extensible, by fully separating SAML message syntax from transport mechanisms.

2.4.1 SAML Core

The SAML Core[18] specification defines a XML based markup language for SAML messages, independently from the underlying transport protocols. As well as the rules for processing each of those assertions.

It is the SAML Core specification that defines which kind of identifiers are suitable to name entities during message transactions, this includes the identifiers used to refer to a users in messages exchanged between the Identity Provider and the Service Provider. The specification defines three main kinds of identifiers:

- Persistent Identifiers
- Transient Identifiers
- Other(Legacy Identifiers)

2.4.1.1 Persistent Identifiers

A persistent identifier is an opaque identifier specific to a Service Provider and Identity Provider, these identifiers are usually constructed using pseudo-random numbers and have no discernable relation with the user's Identity they represent. Since each of these identifiers is only meaningfull between one IdP and one SP, identity correlation across different Service Providers using these identifiers is not possible. This means SAML's Persistent Identifiers are essentially privacy enabled pseudonyms, intended to prevent Service Providers to correlate user information based on SAML identifiers.

2.4.1.2 Transient Identifiers

Transient or one-time Identifiers, are a type of opaque identifier that are only valid during a specific session. This allows the user to make multiple uncorrelated visits to the same Service Provider. Since the used identifiers will be different at each user visit, this identifier can not be used by the service provider to correlate user visits.

2.4.1.3 Other Identifiers

SAML allows for any kind of identifier to be used, it can even be tailored to use identifiers used by an already existing application. Additionally to persistent and transient identifiers, SAML 2.0 defines the following types of identifiers:

- Unspecified
- Email Address
- X.509 Subject Name
- Windows Domain Qualified Name
- Kerberos Principal Name

2.4.2 SAML Bindings

The SAML Bindings[17] specification defines mappings between the messages defined in SAML Core, and specific messages for particular message transport protocols. The current SAML Bindings contains mappings for the following transport protocols:

- SOAP
- Reverse SOAP
- HTTP Redirect
- HTTP POST
- HTTP Artifact
- SAML URI

Each one of which defines the use of SAML in different protocols. For one to extend SAML to be supported on a protocol other than HTTP, one would need to define additional SAML bindings.

2.4.3 SAML Profiles

The SAML Profiles[30] specification defines workflows to use SAML in different scenarios. The current SAML Profiles specification defines profiles for:

- Identity Provider Discovery
- Single Sign On
- Single Logout
- Artifact Resolution
- Name Identifier Mapping
- SAML Attribute

2.4.3.1 Identity Provider Discovery Profile

This profile describes the means through which a Service Provider may determine which Identity Provider to use, in order to authenticate a user. SAML only proposes one profile for Identity Provider Discovery that uses HTTP cookies to hold the URI of the Identity Provider. The proposed solution however only works for scenarios were only one Identity Provider exists and requires that both the Identity Provider and the Service Provider share a common DNS domain.

SAML does not possess a proposal for a method that allows a Service Provider to determine the user's Identity Provider in scenarios involving more than one Identity Provider. However in scenarios where the User Agent has support for SAML authentication this is not necessary.

2.4.3.2 Single Sign On Profiles

SAML 2.0 defines two different profiles[30] for Single Sign On, one for plain web browsers and one for enhanced clients with embedded SAML support.

In order to support SAML based SSO for users with web browsers with no SAML support, SAML proposes a binding that works in a similar fashion to the OpenID workflow seen previously in Fig. 2.8, this profile uses HTTP REDIRECT messages to redirect the user's User Agent from the Service Provider to the Identity Provider(fig. 2.12). It must be noted that step 2 in fig. 2.12 assumes the Service Provider can determine the appropriate Identity Provider to redirect the user to. As seen in 2.4.3.1 SAML has limited support for discovery of Identity Providers.

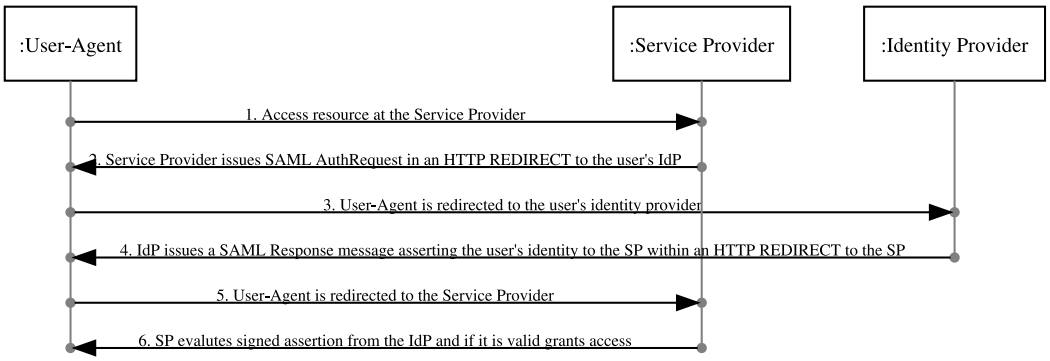

Figure 2.12: SAML SSO workflow for web clients with no support for SAML

The second SSO profile defined in SAML is called ECP(Enhanced Client or Proxy) and is intended for entities that support SAML through SOAP messages. While in the previous profile the User Agent was redirected between the SP and the IdP, now the User Agent actively initiates all the necessary connections, with either the Identity Provider or the Service Provider(fig. 2.13).

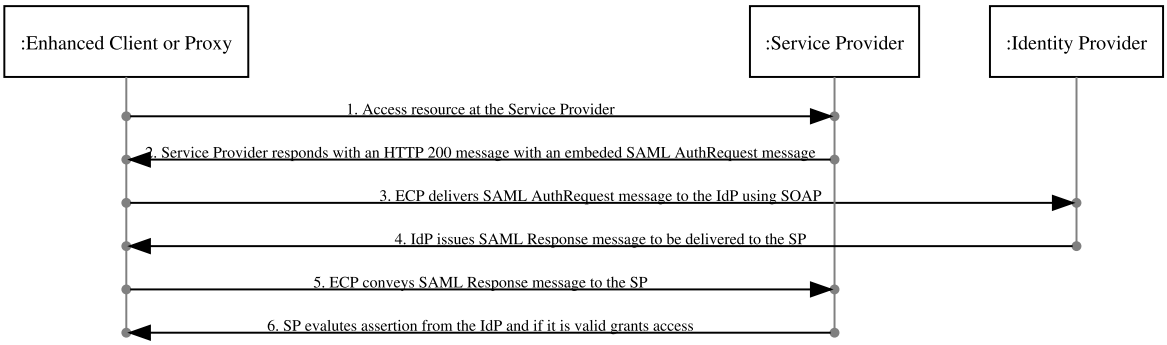

Figure 2.13: SAML SSO workflow for clients with support for SAML

Communication between the User-Agent and the Service Provider is done using regular HTTP messages, while communication between the User-Agent and the Identity Provider is done using SOAP. It is assumed that the User-Agent implicitly knows which Identity Provider to contact.

This second workflow has several security and privacy advantages. While the use of HTTP REDIRECT seen in the first workflow allowed for support for clients with no support for SAML, it made the user vulnerable to phishing attacks since a malicious Service Provider could redirect the user's browser to a fake Identity Provider, that would collect the user's authentication credentials. Additionally in the first workflow the Identity Provider had to know from which Service Provider the user was coming from, in order to be able to redirect the user's User-Agent to the appropriate Service Provider, thus implicitly enabling the Identity Provider to keep track of the Service Providers the user visited. In sum this workflow allows for phishing resistant authentication and increased privacy management.

2.4.3.3 Single Logout Profile

The Single Logout Protocol is a SAML mechanism that can be used by a user to logout a from existing active sessions in all Service Providers the user is federated with a particular identity. The logout operation could be initiated by user or by the Identity Provider alike and could refer to one single session(one Service Provider), or multiple sessions.

2.4.3.4 Artifact Resolution Profile

SAML supports the concept of references(called Artifacts) in its messages. Artifact Resolution is the process of dereferencing an Artifact into the corresponding SAML message. An Identity Provider could for example use Artifacts over an insecure channel, while ensuring that Artifact resolution was always done using a secure channel, the artifacts are just an identifier with meaning only for the entity that issued it, an attacker would gain nothing by eavesdropping on messages carrying only artifacts.

2.4.3.5 Name Identifier Mapping Profile

This profile provides a way to map different identifiers used by the same user. For example if two Service Providers wish to exchange messages about the same user, but the user has federated with each Service Provide using different Persistent identifiers for each. If the user consents to this exchange of information between the providers, the Identity Provider could map a new identifier that both Service Providers would use to refer to the user.

This is typically used in complex scenarios, where different Service Providers cooperate, in order to provide the user with added value services.

2.4.3.6 Attribute Profile

The SAML attribute profile defines the format and naming of attributes. Attributes are typically embedded in assertion messages, to convey additional information about the subject of the assertion. For example an Identity Provider could include the email address of a user, when issuing an assertion about a user.

2.4.4 Security

SAML supports use encryption or signatures to ensure message integrity or authentication when needed. Either through mechanisms associated with the communication channel(TLS could guarantee two-way authentication, integrity and confidentiality), or through encryption or signing of SAML messages using XML-Enc [23] and XML-Sig[24] respectively. This allow granular signing

or encryption of SAML messages, an Identity Provider could choose only to encrypt sensitive information, while leaving the remaining information in clear text.

The SAML workflows seen earlier does not refer how the Service Provider verifies the assertions from the Identity Provider. In fact SAML does not specify the necessary trust mechanisms, used to define and convey keys between parties. It assumes that some sort of key infrastructure such as PKI is already in place.

2.5 Information Cards and the Identity Metasystem

Kim Cameron's Laws of Identity[15], outline the need for an "unifying identity metasystem", a system that can abstract implementation details under a unified interface that allows loosely coupled identity management.

It is unlikely that an universal identity system will ever exist. Different contexts will have different requirements for the identity system, and each context will employ the most suitable identity system for the job. The idea behind the identity metasystem, is to provide user with simple and consistent experience while managing their digital identities. While allowing to use any combination of identity provider/technology that the user wishes.

The identity metasystem requires:

- A consistent user experience, across contexts and identity technologies
- A way to represent identities and claims
- Encapsulation formats that allow Service Providers, Identity Providers and Users to communicate in a secure manner, abstracting underlying identity technology

2.5.1 The Information Card metaphor

The Information Card is a metaphor for identity representation. This metaphor represents each identity as a single card, not unlike credit cards or identification cards in a wallet. Each card possesses a visual representation with a card-like picture and a name describing the underlying identity. This metaphor is part of the requirements for a consistent user experience, in the identity metasystem.

Information Cards can be of two different types:

- Managed Information Cards
- Self issued information cards

2.5.1.1 Managed Information Cards

Managed Information Cards are cards issued by an Identity Provider upon user request. Each card refers to an identity managed by a specific Identity Provider. The card itself does not contain any user's personal information, only information about the Identity Provider that manages the Identity. Typically a card will have a list of claims the user identity can provide, as well as a list of assertion types the Identity Provider Supports(SAML 1.0, SAML 2.0, other).

One concern regarding the federated identity model, is that the Identity Provider can track the Service Providers the user visits. Based on this concern Managed Information cards can be classified in three type:

- **Auditing** if the identity of the Service Provider must be disclosed to the Identity Provider
- **Non-Auditing** if the identity of the Service Provider will never be disclosed to the Identity Provider
- **Auditing-optional** if disclosure of the identity of the Service Provider to the Identity Provider is optional

2.5.1.2 Self issued Information Cards

Self-issued information cards are issued by the user himself. For self-issued information cards, the user can be seen as being his own Identity Provider. Since the user issues these cards, they contain all user information related to the identity they represent, as well as all the necessary information to make assertions about the identity they represent to Service Providers. Internally self issued information cards use self issued SAML assertions.

2.5.2 Building the Identity Metasystem

The WS-* protocol family already extensively defines protocols for exchanging messages in the federated identity model(WS-Federated) in a secure manner, using web services. Since these protocols already provided all the mechanisms for an agnostic implementation using other identity systems, Microsoft's initial identity metasystem implementation(Cardspace) was built on top of these protocols.

The WS-Trust[40] is used to represent claims, while the WS-SecurityPolicy[39] and WS-MetadataExchange[22] protocols are used for negotiation. As long as all parties(SP, IdP and User) implement support for these protocols, the underlying identity technology(SAML, OpenId) can be abstracted.

2.5.2.1 Information Card Support

For a Service Provider to support the Identity metasystem it must expose appropriate WS-Trust web services and announce support for information cards, by including appropriate information in its web site. Figure 2.14 shows an XHTML example a Service Provider could embed in a web page, to announce support for information cards. The Service Provider may impose certain requirements to the type of identities user to authenticate. In the presented example, the Service Provider requests that the user must present an SAML 1.0 assertion in order to authenticate and that the selected identity must disclosure a email address to the Service Provider.

```
<ic:informationCard name='xmlToken'
    style='behavior:url(#default#informationCard)'
    issuer="http://schemas.xmlsoap.org/ws/2005/05/identity/issuer/self"
    tokenType="urn:oasis:names:tc:SAML:1.0:assertion">
    <ic:add claimType=
        "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
        optional="false" />
</ic:informationCard>
```

Figure 2.14: Embedding a requirement for information card authentication in a web page

Additional information could be placed within this description. The Service Provider could for example place a link to a privacy policy , for the Identity Selector to present to the user.

Similarly Identity Providers must support the necessary WS-* webservices, so that clients can access their identity information. Essentialy they provide a translation service from the underlying Identity technology to the appropriate webservices.

For users to benefit from the identity metasystem, a special software component called an *Identity Selector* must be installed in the user's terminal. The identity selector provides the user frontend to the Information Card metaphor. Microsoft's Cardspace Identity Selector(fig. 2.15) was the first implementation for an Identity Selector and as time passed other implementation for other systems became available, namely Project Bandit's[2] DigitalMe identity selector and Project Higgins's[6] web based identity selectors.

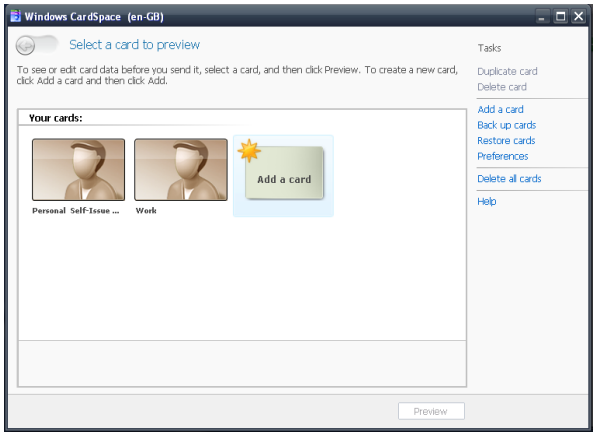

Figure 2.15: Cardspace Identity Selector

2.5.3 Information Card Authentication Workflow

The authentication workflow(fig. 2.16) using Information Cards, is mostly identical to the authentication workflow seen previously for SAML enhanced clients(fig. 2.13). SAML assertion were in fact the first type of assertions to be used with Microsoft's Cardspace. implementation. While SAML 1.0 and 2.0 can be used with the identity metasystem, a specification draft[29] for embedding OpenID assertions in information cards already exists.

Allowing users with OpenID based identities to use information cards, not only brings the OpenID under the Identity Metasystem abstraction but also solves some security concerns regarding OpenID. Because this authentication workflow does not rely on HTTP REDIRECT messages that could make the user vulnerable to phishing attacks.

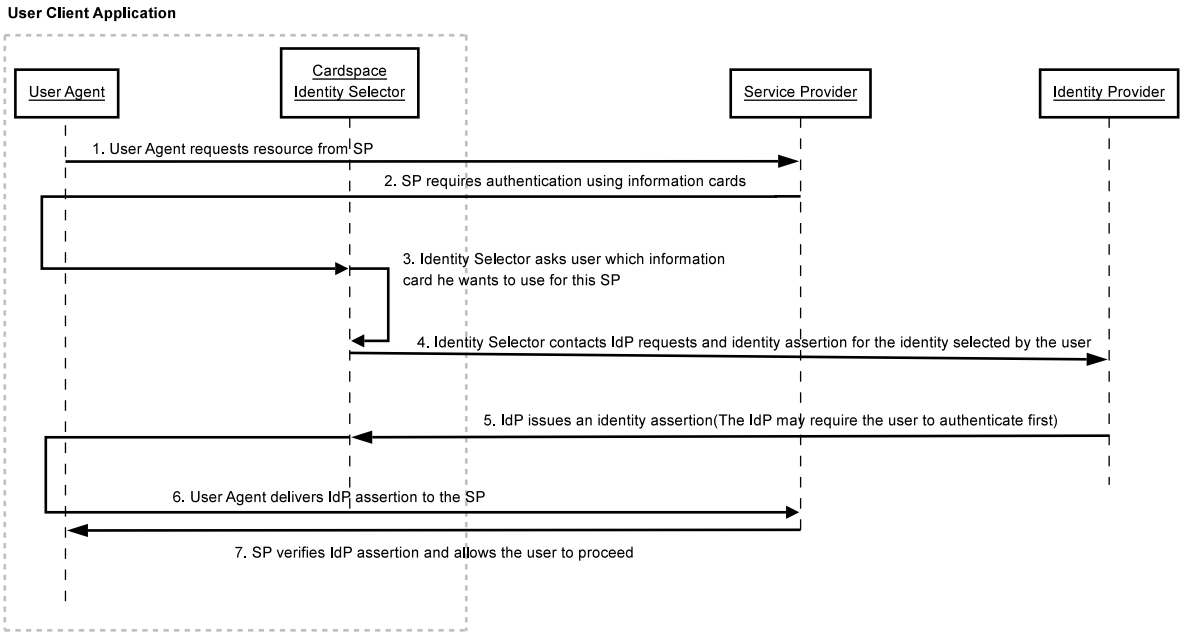

Figure 2.16: Authentication workflow for Cardspace

2.6 Summary and Conclusions

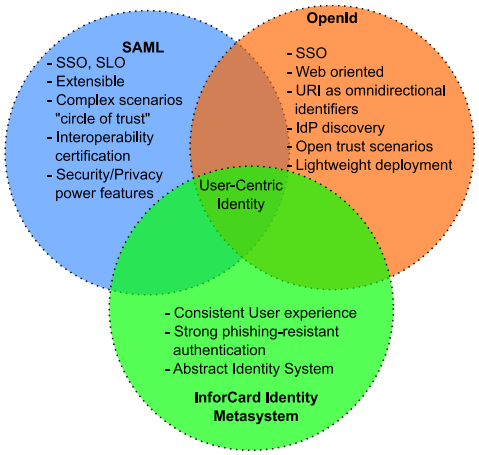

Figure 2.17: Identity interoperability

No single identity solution will be sufficient for everyone. Since the different identity frameworks aim at different identity scenarios, and support different features(table. 2.1), most involved parties now seem to be aiming to an interoperability scenario(fig. 2.17).

OpenID seems to cater the needs of the "open internet", providing Single Sign On and easy omnidirectional identity for the masses. Enabling recognition of the same identity across different

Service Providers, being specially tailored for the "social web" where the user's prime concern it to advertise their identity. SAML on the other hand addresses corporate identity needs. It was designed from the beginning to be extensible, it supports complex trust relationships between a user and multiple Service Providers, while it still has support for identity identifiers from legacy applications. The Identity Metasystem provides a common infrastructure over different identity technologies, taking the first steps to create a user metaphor for identity management.

Both OpenID and SAML have security issues with the SSO mechanisms that employ HTTP REDIRECT messages. Since these rely on the Service Provider to direct the user to the appropriate Identity Provider, a malicious Service Provider could impersonate the user's Identity Provider and steal the user's credentials. These issues can however be mitigated by enabling client support for these frameworks.

	OpenID	SAML 2.0
IdP discovery	Yes using URI/XRI/Yadis based resolution	Limited. Hardcode in SP or using cookies for SPs in the same domain
Identifier Schemes	globally unique URI based identifiers as well as privacy preserving identifiers	privacy preserving identifiers(pseudonyms) and opaque legacy identifiers
Extensibility	Limited. Only on top of HTTP messages	by design

Table 2.1: Feature comparison between SAML and OpenID

Even though these solutions address the needs for federated identity for the web, none of them addresses federated identity outside of HTTP based applications, little or no attention as been given to other protocols. SAML could probably be extended to operate over other protocols, OpenID however is tightly tied to the HTTP protocol. Proposals already exist for use of the SAML infrastructure in SIP[28, 12, 47] and for authentication in network handhovers[33]. While HTTP services are a big part of today's web, SIP also plays a big role in multimedia based services and it would benefit greatly from the federated identity model.

The previously discussed solutions acknowledge the need to use different identifiers for different user identities, as well as control over the dissemination of these identifiers. However none of them address any of identifiers used by the underlying layers. The SAML specification clearly states that, while SAML provides measures to prevent identity correlation using SAML identifiers "correlation may be possible through non-SAML handles"[44].

Chapter 3

Privacy Attacks

In this chapter an attacker model will be presented that outlines the kind of attacks that can be used to threat user identity privacy and the relevant scenarios for these attacks. The objective of the attacker in such a situation, is to gather as much private information about the users as possible and to make correlations about the user's identities based on the collected information.

After defining the attacker model, concrete attacks and some demonstration implementations will be presented for each of the relevant attackers. The methods and implementations presented here focus on web-based scenarios, where the Service Provider hosts its services using web based applications and the user's user agent application is a web browser. Such attacks do not address collection of user personal information, like email addresses or phone numbers since the user usually supplies this information on a voluntary basis, instead they focus on collection of user identifiers and usage of these identifiers for linkage of user identities.

3.1 Attacker Model

The following attacker model outlines the relevant scenarios considered for this work, in order to identity the scope of attacks and consequently the required mitigation measures that must be adopted.

3.1.1 Assumptions

For the attacks considered the following assumptions apply:

- The user does not employ any kind of mechanism that conceals L2, L3 or L7 identifiers within his messages.
- A mobility scenario is assumed, where the user uses the same terminal as he moves across different access networks.
- The L3 identifiers dynamically assigned to the User's terminal are globally unique, which means that no Network Address Translation(NAT) mechanisms are in place
- The only attacks that are considered are the ones where the target is the user

3.1.2 Domain and Assets

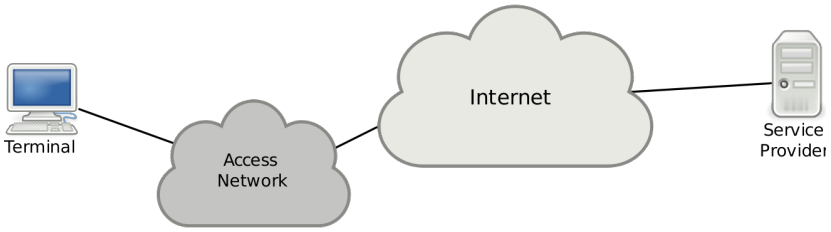

Figure 3.1: Relevant domains for this attacker model

For this attacker model, three domains can be considered:

- Terminal
- Access Network(AN)
- Service Provider

The attacker will attempt to gather user traffic and use the identifiers embedded in the user's messages in order to build profiling information about users, thus profilling user habits, the services the user uses, or how the user moves across access networks.

3.1.2.1 Terminal

The terminal the user employs when connecting to the network, it is assumed the terminal travels with user and contains the following assets:

- L2 Identifier
- L3 Identifier(temporarily assigned by the Access Network)
- Contextual information shared with each Service Provider (L7 identifiers)

3.1.2.2 Access Network

The network to which the User associates his terminal, in order to gain connectivity to one or more Service Providers. It is the Access Network that provides an L3 address to the user's terminal. The user may use multiple access networks.

3.1.2.3 Service Provider

A Service Provider is an entity that supplies a service to a User, it is assumed that any interaction between the Service Provider and the User are initiated by the User. In some cases the User might be registered with the Service Provider for subscription based services.

3.1.3 Attacker Matrix

The attacker matrix(Table 3.1) identifies relevent scenarios for this attacker model. Scenarios that are considered to be out of the scope of this attacker model are marked with a gray background.

		Attacker		
		Access Network	Service Provider	Terminal
Target	Terminal	Can use L2, L3 and L7 identifiers in User messages to correlate User identities. And associates a User with his Service Providers.	Can use L3 and L7 identifiers in User message to link User identities	
	Service Provider			
	Access Network			

Table 3.1: Attacker Matrix

3.2 Attacker: Service Provider

While L2 and L3 identifiers in IP networks are globally unique and easily recognizable, L7 identifiers are specific to the protocol in use, and may even differ among Service Providers using the same protocol. Using L7 identifiers the Service Provider can associate different visits by the same User. However they tend to be volatile in nature, for example when using the HTTP protocol, the User can clean the client application caches disposing of all L7 identifiers.

In such situations where L7 identifiers are volatile, the Service Provider may make use of L3 identifiers to associate different user visits. While L3 identifiers are globally unique, they are dynamically assigned to users and may change along the time. The Service Provider should only consider correlation of User visits from the same IP address if they happened within a reduced time frame. An attacker Service Provider could combine the previously described measures, in order to maximize the probability of successfull User identity correlation.

Since the Service Provider can user either L3 or L7 identifiers in order to correlate user visits, for a User to be able to hold multiple unlinkable identities while accessing a Service Provider, each

identity must employ distinct L3 and L7 identifiers.

3.2.1 Cookies as user unique identifiers

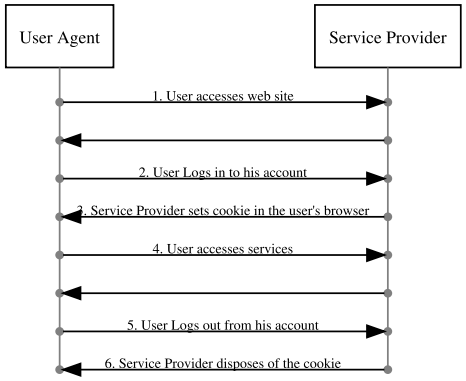

Figure 3.2: Desirable use of cookies by a Service Provider

Web Cookies[35] are a mechanism to create statefull sessions between an HTTP client and an HTTP server. Essentially a cookie is piece information generated by the web server and stored in the user's browser. When a web server sets a cookie in the user's browser, the browser will send the cookie back to the web server each time it connects to the server.

Cookies were initially intended to provide web servers with ability to recognize previously authenticated users(fig. 3.2). So that the web server after authenticating a user would set a cookie in the user's browser in order to recognize the user in posterior message exchanges. However there is no restriction that prevents a web server to set a cookie in the user's browser at any time(3.3). While cookies themselves do not uniquely identify a user, the information placed in cookies can uniquely identify a user. Typically web servers place some sort of unique identifier in the cookie, thus such identifiers are only meaningfull for the web server that generated the identifiers.

Each cookie typically has an age limit, the amount of time after which the user's browser will dispose of the cookie. Some browsers do dispose of cookies with no age limit as soon as the browser closes, but web servers are known to set large cookie age limits(up to several years). Since by default browsers do not clean these cookies, one could envision a scenario where a cookie is stored indefinitely at the user's browser, thus making the user permanently recognizable by the Service Provider that generated the cookie.

The user could disable the browser's cookie support, or at least block cookies by default. However service providers are know to require cookie support when using their web sites, furthermore work presented in [25] introduces "cache cookies", a method to place unique identifiers within the browser's cache instead of using traditional cookies.

For a user to prevent a Service Provider from injecting unique identifiers in the user's browser,

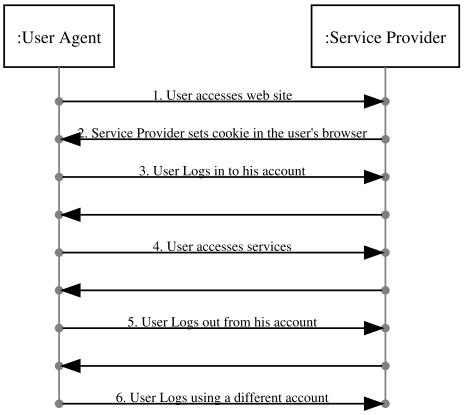

Figure 3.3: Use of cookies by a malicious Service Provider

the user would have to:

- Disable cookie support
- Disable the browser's cache

The lack of caching would result in severe performance degradation, and the lack of cookie support would prevent the user from using the majority of modern web sites. This makes disabling of these technologies an impractical tradeoff.

3.2.2 Exploiting the Browser Cache

Work developed in [25, 32] reveals that, web browsers are especially vulnerable to release of user private information to Service Providers through its caching mechanisms. Modern browsers keep two kinds of caches created from the pages visited by the user:

- Page cache with the contents of previously seen pages
- Visited links cache with the URIs of links the user has previously visited

If an attacker were to have access to these caches, he could determine which sites were visited previously using that particular browser. While not being able to determine the full contents of these caches, an attacker may be able to determine whether or not a certain object(URI) is already present in cache. Web browsers behavior will change depending on the contents of its cache and an attacker could generate content that would cause the browser to react in a specific manner if an object is already in cache.

Attacks based on this concept allow the attacker to query the browser's cache for a specific object. However such attacks don't usually scale well, since for a large number of queries they place a large workload on the user's browser making them noticeable.

3.2.2.1 Page Cache

The page cache typically stores the actual content of web pages. Html, images and other types of content are placed in this cache in order to reduce page load times. Each time the browser needs to fetch an element from the network, it checks the cache previously and if the object is not available in the cache it actually transfers it from the web server.

If an attacker Service Provider were able to determine if a specific URI was already loaded into the browser's page cache, the attacker could determine if the user visited that URI prior to visiting the attacker's web site. The work in [25] already presents several attack methods an attacker can use to query the user's browser cache. Simply put, an attacker measures(fig. 3.4) the time it takes for the browser to fetch several resources and compares the times, in order to determine which resources were already in the cache.

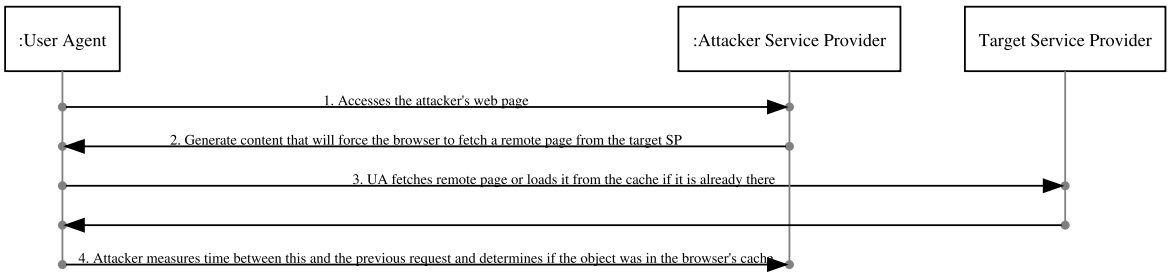

Figure 3.4: Attacker SP determining if a target page is in the browser's page cache

This would effectively allow an attacker to query the browser's cache for specific content. The attacker could for example query the cache for a specific list of sites. If this list were to be large enough it might be used to identity a specific user, effectively using the user's behavior as an identifier.

3.2.2.2 Visited Links Cache

Visited links cache are kept by browsers in order to maintain a registry web sites previously visited. This cache serves two purposes:

- Provide the user with a history of visited sites
- Provide visual differentiation between visited and unvisited links when rendering a web page

The former purpose is the most relevant for attacks based on this cache. Most browsers render previously visited links in a web page in different manner, usually employing a different color for those links. A web site can generate content that causes the browser to react differently according to the type link seen in a page. A malicious Service Provider could include javascript code in their

web page, that would verify if the links in a web page were marked as visited or not and convey this information back to the Service Provider.

Privacy exploitation based on querying of the visited links cache is a known privacy risk. Firefox developers have extensive discussions[3] on this matter, as well as multiple implementations of this kind of exploits. However general consensus seems to be that this issue can not be dealt with without breaking browser functionality.

Attacks based on this cache will yield similar results, to the ones based on the page cache. However implementations for these attacks are considerably simpler, than the ones used for the page cache. Additionally the visited links cache will probably hold more entries than the page cache, the page cache is usually constrained by size on disk while the links cache is usually constrained by number of objects or by lifetime.

For example the Firefox browser holds a page cache with a maximum size of 50Mb and a visited links cache with a maximum entry lifetime of 90 days. In contrast the Opera web browser holds a page cache with a maximum size of 20Mb and a visited links cache with a limit of 500 objects. These value are configurable, but the presented values are the default for these implementations.

3.2.3 IP addresses as user geographic locators

Some Service Providers attempt to determine the user's geographic location from the user's IP address. This is done by examining the user's IP address and determining to what country and organization the IP address is assigned to. Such Service Provider usually hold a database that maps IP address blocks to organizations, countries and even geographic coordinates. Such use of the user's IP address is quite controversial, since users are unable to prevent the Service Provider from determining the user's location in this manner.

While such databases will usually have a correct mapping from the IP to country and organization. The accuracy of such databases may vary, public databases can sometimes be quite inaccurate while some commercial services that offer location from IP are extremely accurate.

Service Providers seem to have multiple uses for IP addresses as geographic locators:

- Location based Services
- Location based access control
- Location based content

3.2.3.1 Location based Services

Some Service Providers adapt the service offer according to the user's location, for example providing directions for the nearest retailer from the user's location. Or directing advertisement for user's in a specific location.

3.2.3.2 Location based access control

Service Providers also seem to employ the geographic information they acquire from the user's IP address, as a parameter for access control to specific services. Some Providers that wish to restrict their service to a specific country, usually determine if the user's IP is attributed to that country, prior to allowing service.

For example Service Providers that broadcast online movies, are legally constrained to only broadcast movies to users within their licensed geographic area. Broadcasting of video on the internet under such constraints, requires the Service Providers to make sure that they only allow clients with an IP address assigned to their geographic area.

3.2.3.3 Location based content

Location based content, is content tailored for users in a specific locations. For example providing content in the user's language, or adapting the layout of a web page to suit geographic usability features. After all there is extensive information related to content presentation, that can be used by determining the user's locale:

- Numeric formats
- Date and time formats
- Measurement units(metric or imperial)

This kind of use for the IP address is rather debatable, specially for web sites, since the HTTP protocol already has a header[13] for the user to explicitly specify the desired language and country locale. This seems preferable, since it will not victimize traveling users, who find themselves in a foreign country trying to the understand a language a Service Provider as erroneously attributed to them.

3.2.4 Implementing a Service Provider User information collector

Files:
identitysite/addsite.php
identitysite/sitelist.php
identitysite/index.php
identitysite/jsprofiling.php
identitysite/include/common.php
identitysite/include/startup.php
identitysite/openidlist.php
identitysite/postsites.php
identitysite/include/settings.php

For the purpose of demonstrating an attack by a Service Provider, a web application was developed during this work that applies some of the attacks described previously. This implementation:

1. Immediately sets cookies in the user's browser
2. Mantains a list of web site URIs that will be used to query the browser's visited links cache
3. Attempts to correlated different user visits either using cookies or the user's IP address
4. Stores all the collected information in a database
5. Allows the user to authenticate using OpenID

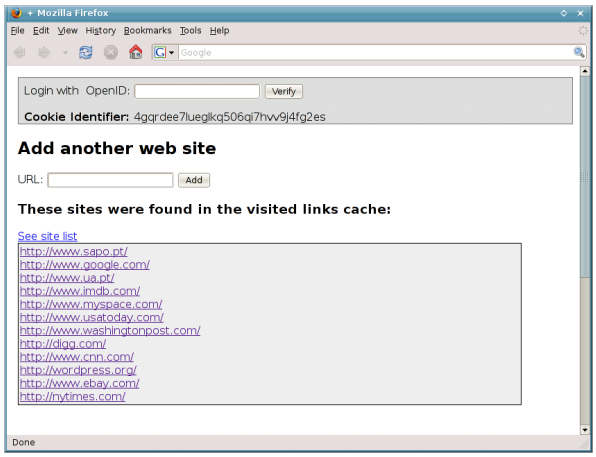

Figure 3.5: Service Provider implementation showing information collected after one visit.

This implementation was accomplished using the PHP programming language and the storage of user information is done in a sqlite[9] database. The methods involving IP as geolocation identifier and the browser's page cache were not introduced is this implementation. The use of IP addresses as geolocation identifiers is hard to demonstrate in small testing environments and the attacks based on the visited links cache, already provide the attacker with most of the information he could acquire from analogous attacks based on the page cache.

For each user visit, this web application verifies if the user' browser already has a cookie set. If a cookie is not set, it defines one with an age limit of 10 years. Internally the web application has a list of URIs to be matched against the browser's visited links cache. The main web page has a javascript script embedded, that will query the browser's cache and send the collected information back to the server.

Subsequent user visits are tracked using either the cookie that was set previously, or the user's IP address. Since the user's IP address may change with time, only visits by the same IP address in a 1 hour time window are considered to be from the same user.

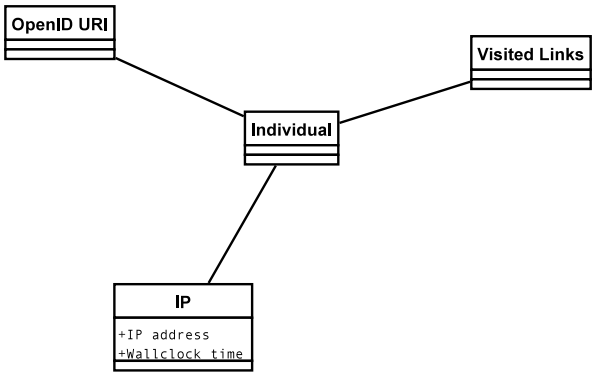

Figure 3.6: Information model stored by SP implementation

This is implementation was initially intended as a tool for study of OpenID and as such it supports OpenID authentication. Since OpenID didn't have any relevant flaws to be exploited, the developed attacks do not exploit any OpenID specific issues but rely instead on other mechanisms. This emphasizes the point that OpenId's(or other identity framework) detachment from the lower layers makes the user vulnerable to identity correlation using non-OpenID identifiers.

3.3 Attacker: Access Network

At the access network the attacker can eavesdrop on all the User's traffic, L2 identifiers can be used to identify the User terminal across visits. The attacker is able to eavesdrop all User traffic and record:

- User L2 identifier
- L3 identifier of the Service Providers the User contacts
- Wallclock time when the traffic was captured

Since the L2 identifier in the User's terminal is unique, the attacker would then be able to know what services a specific terminal uses regardless of the identity the user was fullfilling. In the access network L2 identifiers are enough to correlate different accesses by the same user, L3 identifiers on the other hand are not so relevant since they are attributed by the access network.

Initially the use of L7 identifiers by an attacker in the Access Network was dismissed. Since L7 identifier semantics vary between protocols and service providers, the use of these identifiers seemed to be restricted to the Service Provider. While L2 and L3 identifiers have clearly defined semantics such is not true to L7 identifiers in web applications. For example the content of a web cookie varies greatly from provider to provider, an attacker could not possibly understand cookie semantics for all existing Service Providers. While this holds true for the majority of the cases, there is at least one exception. Google offer a free service to other Service Providers called Google Analytics[4].

Google Analytics gathers user information from sites and generates profiling results from this data. For this to work each Service Provider places a snippet of javascript code in their respective websites. This particular snippet of code injects some identifiers in the user browser's cookies. This is not uncommon[34], for example advertising companies with advertisements banners inserted in third-party web sites are known to collect user information in this manner. The difference with Google Analytics is that it has been so widely adopted by other Service Providers, that almost all web sites use it.

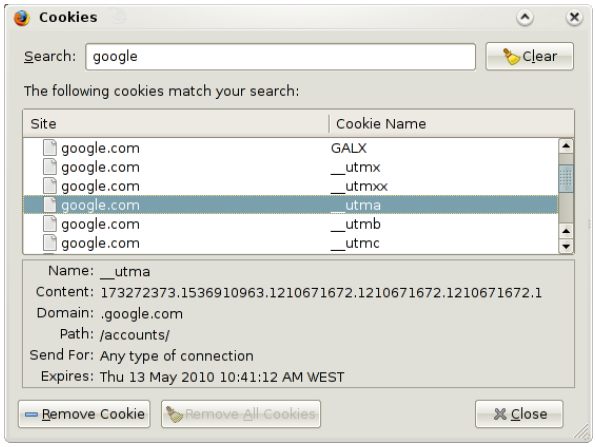

Figure 3.7: Google Analytics identifiers within cookies

Google Analytics places an identifier named **__utma**(fig. 3.7) within the user's cookie, that maps a user to a specific service provider. The identifier placed in this field is a combination of:

- A hash of the Service Provider's web site hostname
- A random number
- The wallclock time from the user's terminal

Google Analytics ended up injecting a common cookie semantics across most Service Providers. Armed with this information an attacker at the access network might consider using these cookies as user identifiers.

Given that the attacker uses L2 identifiers to identify the User, in order for a User to be able to assume multiple non-linkable identities in such a scenario, each identity the User assumes would have to use a different L2 identifier. If each identity made use of a distinct L2 identifier, the Access Network will "see" one terminal per each User identity, where in fact only one terminal exists. The Access Network would then assign one L3 identifier per User identity, instead of one L3 identifier per User.

3.3.1 Implementing an Access Network traffic analyzer

Files: probe/src/monitor.c

There are already several software implementations intended to capture network traffic. Instead of implementing a solution that captures and parses network traffic. This implementation solution uses network traffic already captured by tcpdump[10], processes the traffic and stores user information in a sqlite[9] database. While this implementation could easily be tailored to process traffic in real time, as it arrives from the network(in fact initially, this implementation acted in that manner) the insertion of records into the database under severe traffic loads, was found to be a considerable bottleneck.

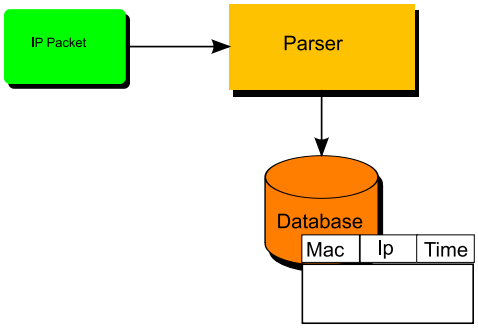

Figure 3.8: A parser implementation to process captured network traffic and store it in a database

This implementation does not store repeated entries with the same IP, Mac and time. Additionally mappings where the Mac address is the broadcast address are discarded. The implementation can be used in the form of a **parser** command, that will read a capture and store information in a sqlite database file. The most relevant methods implemented are:

packet_handler() - The method that parses the IP packets, fetches the relevant identifiers and passes them to other methods for verification or storage.

add_correl() - This method does some sanity checks on the collected information and if it is valid it stores it in the database.

Earlier it was referred that Google Analytics identifiers could also be used for correlation. While exploitation based on these identifiers will not be pursued further in this work an alternative development branch was created that holds the necessary changes so that the implemented parser can find and use these identifiers as a basis for correlation.

3.4 Multiple Attackers

If multiple attackers were to share collected data and cross-reference user information, additional information could be inferred about the User. Table 3.2 presents some information attackers could gain by cross-referencing information. A worst case scenario for attacker cooperation, occurs when the attackers can see the user at all times(all the SPs and ANs would cooperate to attack the user), in this situation it could be considered that an omnipresent attacker exists.

	Access Network	Service Provider
Access Network	Track Users across different access networks, gather user geographic habits	Correlate user location information with Service Provider access, "where is the user when he accesses the service". Or even associate different devices with the same user
Service Provider		Track user across different Service Providers, build profiling information on what other service providers the user uses

Table 3.2: Attacker gains by cross referencing information

This is the scenario were L3 identifiers become the most significant, for correlation of information collected at Service Provider and at the Access Network L3 identifiers are the common information between both information sets. Traffic captured at the Access Network and at the Service Provider with the same L3 source address within a restricted time interval will most likely belong to the same terminal.

If the attackers are able to share information amongst themselves, then all counter-measures referred earlier must be employed simultaneously, it is not enough to guarantee non linkable User identities only in the Access Network or only when accessing Service Providers. To guarantee non linkable User identities each identity must make use of its own distinct set of L2,L3 and L7 identifiers.

3.4.1 Implementing Information Merger

Files: probe/src/merge_db.c

For a scenario with multiple attackers, it is necessary to provide a way to correlate information from the multiple attackers. An implementation is proposed that addresses the correlation of information between a Service Provider and an Access Network. This implementation essentially merges the two databases generated by the previously described implementations, using the user's

IP address and time of access to cross information. The information model that results from merging both databases is presented in fig. 3.9.

The most relevant methods for this implementation are:

merge_individuals() - Merges two individuals, this method is usually called when in the process of merging both the databases, two individuals are found to use the same mac address, in which case all information associated with both of them is merged.

associate_individual_mac() - This method associates a mac address with the remaining information regarding an individual.

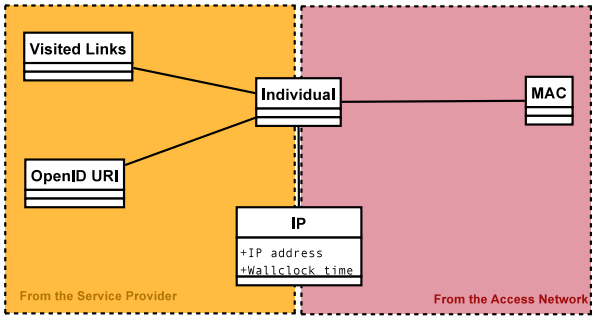

Figure 3.9: User information model that results from the cooperation between an SP and an AN

Chapter 4

Mitigation

The previous chapter presented attacks that used L2, L3 and L7 identifiers to profile user information without the user's consent. Since frameworks like OpenID abstract lower level details they cannot offer protection against such attacks, these attacks can be mitigated through the model specificed in existing bibliography[48, 27] that emphasize that a solution that involves all network layers must be taken. This model however is not enough to guarantee that there is no unintended disclosure of private information, there is one additional precaution that must be taken into consideration regarding the selection of identities when contacting a Service Provider.

In this chapter two implementations are proposed to mitigate the use of the beforementioned attacks for identity correlation. The first is an identity aware web browser, based on the Webkit[11] browser engine, that enables the user to switch identities when he seems necessary and guaranteeing that the user's identities are unlikable by any attacker. The second proposed implementation is an Identity selector for legacy applications, that can constrict legacy applications with no support for identity, by forcing them to employ L2 and L3 identifiers for a specific identity. Both implementations are based around the Virtual Identity Proxy(VIP) that provides distinct L2 and L3 identifiers for different virtual identities The proposed implementation extends the work in the VIP by providing distinct L7 identifiers for different virtual identities as well as "a priori" identity selection.

4.1 Virtual Identity Model

The Identity model seen in [48, 27] envisions a model for Identity Management where each identity has distinct network identifiers for each virtual identity, by segmenting the network stack on a per-identity basis(fig. 4.1). This model introduces Identity Management not only as an horizontal layer on top of the network stack as seen earlier for other approaches but also as a vertical control layer across the other layers. This alone precludes Identity correlation based on network identifiers, since actions under each identity will make use of distinct network identifiers.

As was described earlier interactions with the Service Providers follow a client server model, where the user immediately discloses identity information upon contacting the Service Provider. For the user to have control over the dissemination of his private information, the user must consciously choose the identity he wishes to present to the service prior to contacting the Service

Provider.

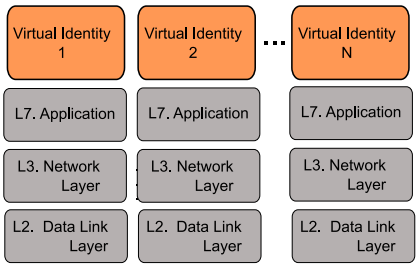

Figure 4.1: Network stack segmentation to provide distinct network identifiers for each identity

Merely by contacting the Service Provider the user exposes some of his identity information, what happens in fact is that the act of contacting the SP forcefully selects(although only partially) the user's identity. Any kind of "a posteriori" identity selection will simply result in further detailing the already selected identity. OpenID provides an "a posteriori" identity selection mechanism through "directed identity", however these mechanisms involve context information defined by the Service Provider that would allow it to correlate different identities even if a per-identity stack segmentation was in place.

Since a posteriori identity selection may result in undesired identity disclosure or in undesired identity correlation, any kind of identity selection must take place prior to contacting the Service Provider.

4.2 Development Environment

All the described components were developed on a Linux environment, using kernel 2.6. Code was developed using the C and C++ programming languages as well as some amount of bash scripting. The proposed implementations are built on top of existing components, namely:

- The Virtual Identity Proxy implementation developed at the Instituto de Telecomunicações - Aveiro
- The WebKit browser engine

4.2.1 Virtual Identity Proxy

The Virtual Identity Proxy(VIP) is an existing implementation that implements the use of distinct L2 and L3 identifiers for different virtual identities. It works by enabling the sharing of a single network interface among several virtual interfaces(VIF). Each VIF possesses a different MAC address, thus appearing as a different network entity for all other network entities. For all purposes from the point of view of an observer in the same network as a terminal using VIP, each VIF will be seen as a distinct terminal on the network.

Communication with the VIP daemon is done using UNIX sockets. Applications can connect to the appropriate socket and register Virtual Identities(VID). In response to a registration, the VIP will create a virtual network interface with a distinct mac address and reply with the name of the interface to be used for that virtual identity. The generated virtual interfaces will be named *vifN*, where *N* is a positive number starting at 0.

Internally the VIP handles all necessary mappings between the real interface the virtual interfaces, by flooding the virtual interfaces with the traffic that arrives to the real interface. For any application in the system a virtual interface created by the VIP is identical to a real network interface. The VIP already ensures distinct MAC addresses for each of the virtual interfaces, since these interfaces can be operated as any regular network interface, an IP address can be acquired as it normally would be, using a DHCP client or by statically assigning an IP address.

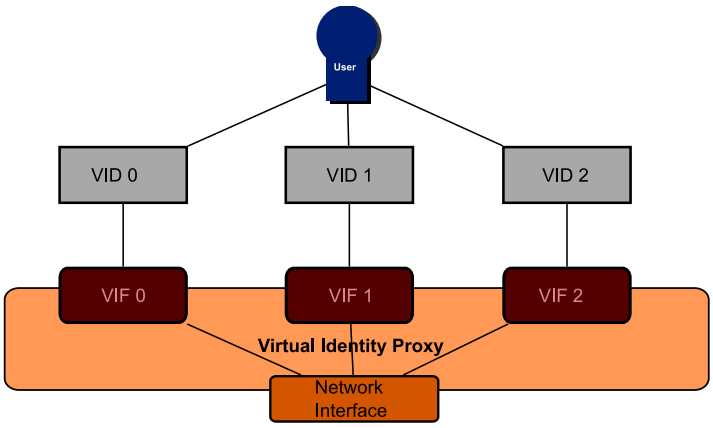

Figure 4.2: The VIP mantains one virtual network interface(VIF) per virtual identity(VID)

4.2.1.1 VIP Daemon

The VIP daemon will attach itself to a specific network interface switching the real network interface to promiscuous mode, so that it can intercept packets destined to the virtual interfaces. Then the daemon will wait for other applications to contact the daemon through existing UNIX sockets. Two sockets are used, one to send messages to the daemon that resides at */tmp/vip-socket* and a second socket were the daemon places responses that resides at */tmp/id-socket*. VIP defines several types of messages, for the purposes of this work the most relevant types are:

- VIP_LOCAL_REGISTER
- VIP_LOCAL_REPLY

The VIP_LOCAL_REGISTER is a message that can be sent to the daemon to register a new interface. The VIP_LOCAL_REPLY message type is usually used to reply to a VIP_LOCAL_REGISTER message and it usually contains the identifier of the newly created interface.

4.2.1.2 Contributions to the VIP implementation

This work as also resulted in some contributions to the VIP implementation:

- Optimizations regarding packet handling by the VIP daemon
- Memory optimizations by reducing the allocated memory

Internally the VIP implementation places the real network interface in promiscuous mode, it then captures all the packets that pass through the real interface and floods the virtual interfaces with the same packets. This behavior is conceptually similar to what happens in a network were multiple terminals are connected to a hub, where the hub acts as a repeater that forwards traffic from one interface to all the others. This behavior can be significantly optimized to reduce the toll on the VIP implementation, since it is the VIP that created all the virtual interfaces it can safely make some L2 based forwarding decisions by forwarding the incoming packets only to the appropriate interface, by comparing the destination L2 address in the packets with the L2 address in each virtual interface. The only exception are multicast and broadcast packets that will continue to be forwarded to all the virtual interfaces.

One other issue that was detected regarded the allocation of memory for forwarding of packets from the real interface to the virtual interfaces. The VIP implementation allocated one memory buffer for each outgoing interface for each packet, this resulted in complexity of *O(n)*(where *n* is the number of virtual interfaces) allocations and deallocations for each incoming packet. This behavior was improved so that the same buffer can be used for each packet.resulting in a *O(1)* complexity regarding packet buffer allocation.

While working with VIP implementation some conclusions were taken regarding the use of the virtual interfaces by other applications. The first being that the virtual interfaces created by VIP have issues with the use of the IPv4 protocol, the virtual interfaces seems to drop incoming IPv4 traffic for an unknown reason. This work as well as prior work with VIP makes use of IPv6 only and thus disregards usage of VIP with IPv4, since usage of IPv4 is not essential for this work solving of this issue is deferred to a later opportunity being noted however that this issue seems to revolve around some sort of issue with the ARP protocol since entries in the ARP table are not created for ARP messages arriving in the virtual interface. There are other virtualization related implementations that suffer from similar issues, but since this is not crucial for this work this issue was not approached further.

4.2.2 WebKit

WebKit is an open source web browser engine, used to implement web browsers. It is the basis for multiple web browsers including Apple's Safari and GNOME's epiphany browsers. It is also used in several browsers for mobile phones, like the Nokia S60, and Google's Android platform.

WebKit has a large abstract core implementation that handles platform-independent functionality, like HTML and Javascript handling. Around this core there are multiple ports, that implement

platform-dependant functionality for different platforms. Ports for webkit can implement, low level functionality like network communication or data storage, or high-level functionality like support for native presentation widgets(fig.4.3). Different operating systems will certainly use different ports, for the purposes of this work, all changes will be made at:

- WebKit's core
- libcurl port, for low-level functionality
- gtk port, for presentation functionality

At the time when WebKit was considered for this implementation, the gtk and libcurl ports for WebKit were considered to be the more suitable from the ones available for the Linux operating system. Other ports exist for Linux, however they are still incomplete missing some necessary features for this work.

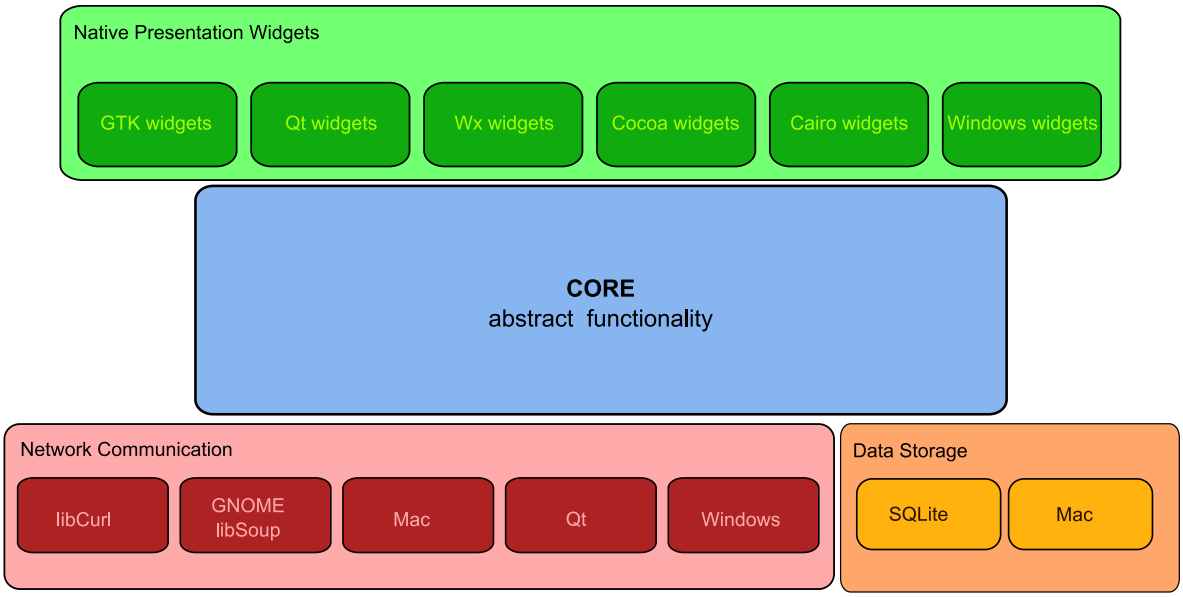

Figure 4.3: WebKit overall architecture. Core functionality surrounded by platform-specific ports.

4.2.2.1 WebKit libcurl port

Libcurl[7], is an C client side URI transfer library, that supports multiple protocols, including HTTP and HTTPS. It also supports SSL certificates and web cookies storage. The libcurl WebKit port, uses libcurl to handle network related communications with remote HTTP(S) servers as well as cookie storage.

4.2.2.2 WebKit gtk port

Gtk[5] is a toolkit for creating graphical user interfaces, written in the C programming language. The WebKit gtk port, adds support for use of gtk widgets in WebKit based applications.

4.3 Identity Aware Browser

To demonstrate how to support unlinkable identities in an application, the WebKit implementation was tailored to support identities based on the VIP. In the previous chapters it was established that in order to have truly unlinkable identities for a browser client, each identity must have:

- A distinct Page Cache
- A distinct Visited Links Cache
- A distinct set of Web Cookies
- A distinct L3 address(IP address)
- A distinct L2 address(MAC address)

The VIP implementation already provides virtual identities, each with a distinct L2 and L3(altough indirectly) identifiers. In order to provide the remaining requirements, support for VIP must be added to WebKit and each VIP identity must be mapped to a one page cache, one visited links cache and one cookie storage(4.4). The concept of cache segmentation for privacy reasons is already explored in [32], while that proposal employs a "same origin" policy for cache separation, this implementation will separate caches according to the user's identities.

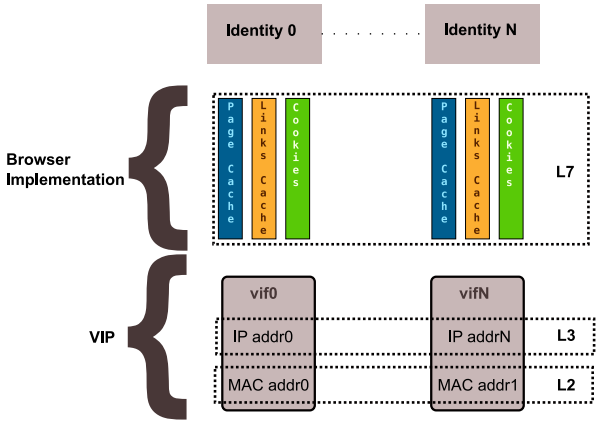

Figure 4.4: Overall implementation

Changes to WebKit's source code were made on top of revision 6dd75b0, from WebKit's git repository located at `git://git.webkit.org/WebKit.git`. The main changes made during this process involved the following components:

- PageCache class
- IdentityManager class
- PageGroup class
- ResourceHandleManager class
- Identity Selector gtk widget
- WebKit based browser implementations

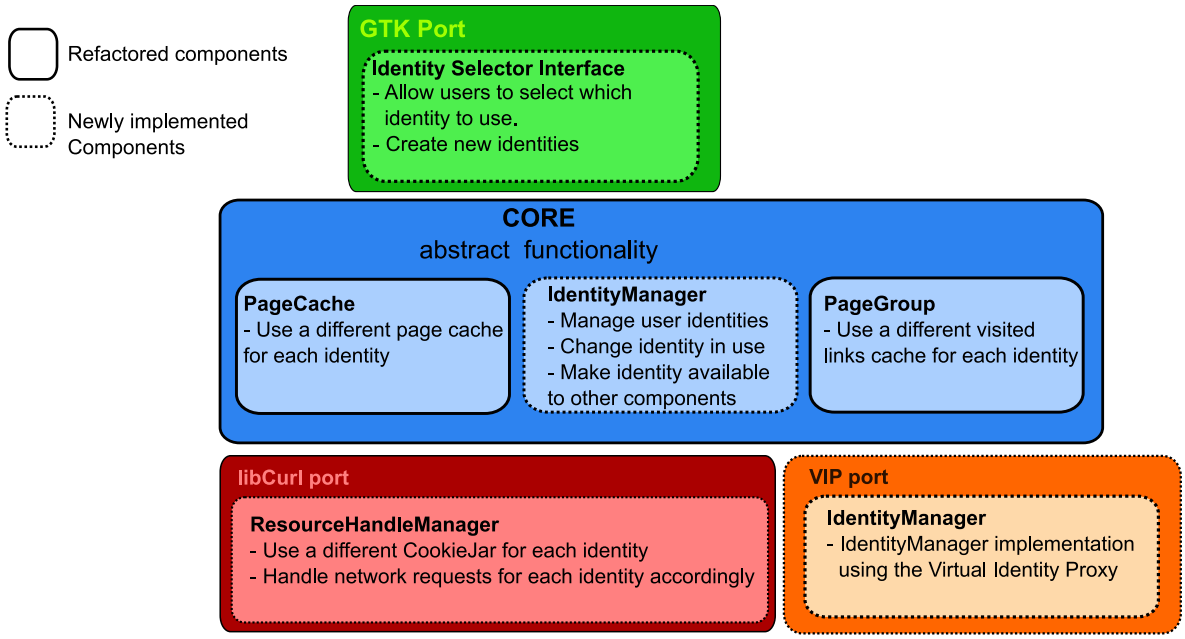

Figure 4.5: WebKit changes, to implement an identity aware browser

4.3.1 IdentityManager class

Files: WebKit/WebCore/platform/network/vip/IdentityManager.cpp
WebKit/WebCore/platform/network/vip/IdentityManager.h

The IdentityManager class introduces support for multiple identities in WebKit. It is this class that interacts with the VIP daemon and manages the various user identities. Internally it keeps two UNIX sockets used to exchange messages with the VIP daemon. Such sockets are opened upon instantiation of the IdentityManager class, which means the VIP daemon must be running prior to launching the browser or the IdentityManager will not be able to communicate with the VIP. This class is located in the WebCore namespace, making it available as WebCore::IdentityManager.

It is this class that provides other components with information about the current selected identity and enables the creation of new identities. Through the IdentityManager class other components can:

- Instantiate new virtual identities
- Query which network interface should be used
- Reset the browser to its default behavior(as if the identity manager didn't exist)

4.3.1.1 Methods

IdentityManager::getInstance() - Since the same IdentityManager must be shared across all other classes, this class is a singleton class and this is the method that can be used, to get an instance of the class.

IdentityManager::getOutIf() - Yields the name of the network interface to be used with the currently selected identity. One constraint that must be considered for this method, is that the returned name is unique for each identity. It is reasonable to assume that other software components will use the returned name to distinguish between identities.

IdentityManager::currentIdentity() - Returns the name of the current identity in use. Since this implementation does not allow the user to name his identities, the name of the network interface to be used with the current identity is returned instead. This is usefull for debugging purposes, but for future implementations one could consider to allow the user to name his identities for usability reasons.

There is always one identity called "legacy" that corresponds to the behavior of the browser prior to this implementation. Under this identity it is undetermined which network interface will be used by the browser to connect to remote servers since that decision will be left to the operating system.

IdentityManager::selectIdentity(iname) - This method can be used to change the currently selected identity, the passed argument is the name of the identity. As seen previously the name of the identity is the name of the VIP virtual network interface.

IdentityManager::reset() - This method resets the IdentityManager to its default behavior, setting the current identity to legacy. This would be equivalent to invoking the selectIdentity() method with the argument "legacy".

IdentityManager::registerIdentity() - Instantiates a new identity with the VIP. This function will block until the new identity is registered or until the method fails. The method provides no feedback whether or not the identity was created. In order to determine if the new identity was created, the list of identities must be checked for changes.

IdentityManager::Iterator - The identity manager provider an iterator to provide listings of the available identities. In the addition to the IdentityManager::Iterator type, the methods IdentityManager::begin() and IdentityManager::end() are also available as is custom for iterators in C++. In order to consult the list of available identities one would transverse the iterator.

```
WebCore::IdentityManager::Iterator it;

for (it = priv->identitymanager->begin();
     it != priv->identitymanager->end(); ++it ) {

     WebCore::String* s = *it;
     /* The string s contains the name of the identity */
}
```

The list of identities will always have one identity, the *legacy* identity. Prior to invoking the selectIdentity() method the contents of this list should be checked first.

4.3.2 PageGroup class

Files: WebKit/WebCore/page/PageGroup.cpp
WebKit/WebCore/page/PageGroup.h

The PageGroup cache implements caching of the previously visited links. All changes made to this class were internal and no additional methods were exposed. Internally this class holds a storage structure for visited links. The class was changed to instantiate a new storage structure for each identity. The name of the identity in use can be fetched from the IdentityManager, this class has only to map each identity to a different storage structure.

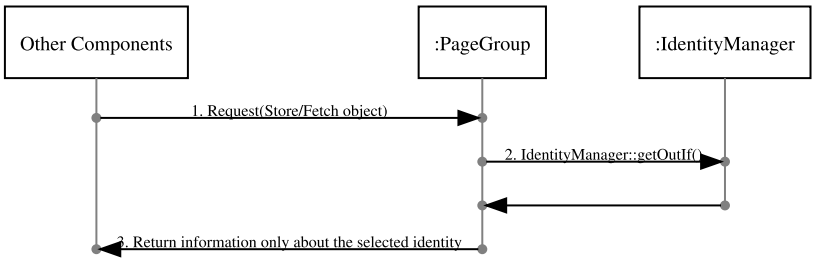

Figure 4.6: Interaction between the PageGroup and the IdentityManager

4.3.2.1 Methods

PageGroup::getVisitedLinks() - This is a private method created to be used internally to instantiate and access the storage structure. The remaining methods rely on this one to provide

them a structure were the links are stored. Changes to this method introduce the use of an hash map to hold different structures for each identity. Each time this method is called, it will call IdentityManager::getOutIf(), it then uses the name of the interface for the current identity as a key to access the appropriate structure. If a storage structure does not exist for a particular identity, this method will create one when it is required.

Since all other methods rely on this one to obtain the storage structure and this method hands out the correct structure for each identity, the interface to the PageGroup class is not broken and other classes that use this class will continue to do so unaware that the user is using multiple identities.

PageGroup::isLinkVisited() - The method that verifies if a given link is already in cache. The only necessary change to this this method was to make use of the PageGroup::getVisitedLinks() method instead of calling the storage structure directly.

PageGroup::addVisitedLink() - The method that adds new links to the cache. The only necessary change to this this method was to make use of the PageGroup::getVisitedLinks() method instead of calling the storage structure directly.

PageGroup::removeVisitedLinks() - The method that removes links from the cache. The only necessary change to this this method was to make use of the PageGroup::getVisitedLinks() method instead of calling the storage structure directly.

4.3.3 PageCache class

Files: WebCore/history/PageCache.cpp

The PageCache class implements storage for the contents of web pages. All changes made to this class were internal and no additional methods were exposed. This class interacts with the IdentityManager class, in order to determine which identity in use.

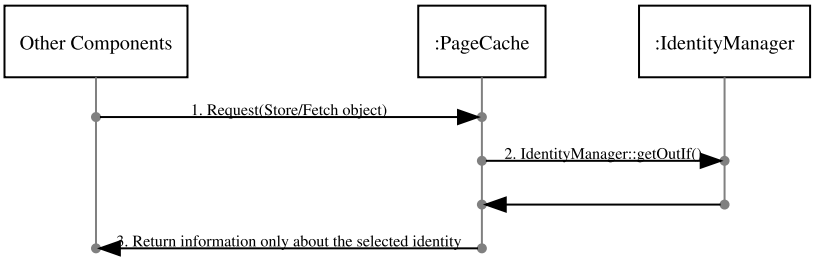

Figure 4.7: Interaction between the PageCache and the IdentityManager

4.3.3.1 Methods

PageCache* pageCache() - The PageCache class has a private constructor, meaning that it is not instantiated directly by other classes. Other components rely instead on the pageCache() method. This method is not a class method, despite being defined in the same file as the class it is standalone method that WebKit provides to access an instance of a PageCache object.

This makes the process of introducing multiple pages caches less complex than the one seen for the PageGroup class. This was the only method that altered in order to provide one page cache per identity. Within this method static mapping structure was introduced to map different PageCache instances to each identity. Each time this method is called it checks in with the IdentityManager class by invoking IdentityManager::getOutIf() and using the interface name as a key for each identity it returns a different PageCache instance depending on which identity is in use.

4.3.4 ResourceHandleManager class

Files: WebKit/WebCore/platform/network/curl/ResourceHandleManager.cpp
WebKit/WebCore/platform/network/curl/ResourceHandleManager.h

The ResourceHandleManager class is part of the libcurl port for WebKit, it handles establishment of network communications as well as storage of cookies. It completely relies on libcurl for all operations related to:

- Handle HTTP requests
- Storage of cookies
- Handling of SSL certificates

It should be kept in mind that this port will probably be subject to changes in the future, other ports already separate cookie storage from network functionality and future work on top of this port should consider that it will probably suffer considerable changes in the future.

The changes made to this class will enforce connections made with one particular identity to be bound to that identity's virtual interface. Cookie storage was also divided in an analogous fashion to what happened with the PageGroup and PageCache classes.

4.3.4.1 Methods

ResourceHandleManager::getCurlSharedHandler() - This method is a private method to the class that was introduced to provide a curl shared handler when needed. The curl shared handler is a structure that is used when invoking libcurl methods. Among other thing it contains information about cookie storage. The main point being that when libcurl handles cookie storage it will have

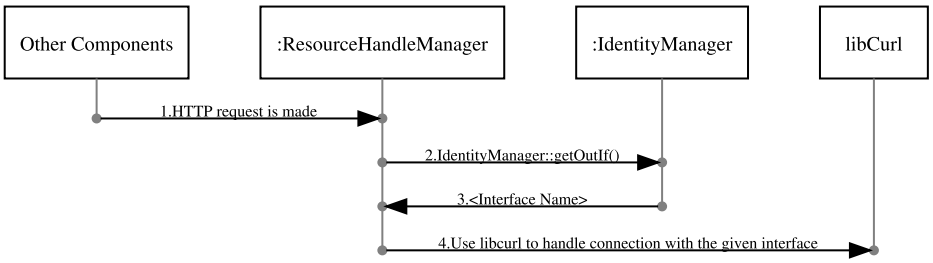

Figure 4.8: Interaction between the ResourceHandleManager and the IdentityManager

different storage locations for each shared handler, in a similar way to what happened with the cache implementation now we will have a different curl shared handler for each identity. This ensures that cookies created under one identity will not be present when under a second identity.

This method will check with the IdentityManager and return a shared handler for the identity in use. If a shared handler for the current identity does done exist one will be created. This guarantees separate cookie storage for each identity. All calls to libcurl from within the ResourceHandleManager class were updated to use this method.

curl_setsockopt_cb() - Libcurl allows the definition of callback functions defined by the caller that will be called on certain situations. One of the definable callbacks for libcurl is a method to operate on sockets that libcurl will use to connect to the remote servers before the connection is made. This method is the callback that was created to bind libcurl sockets to the appropriate interface for the selected identity.

This method gets the name of the network interface to be used from the IdentityManager and binds the socket provided by libcurl to that interface. If for some reason the socket cannot be bound this method will return an error preventing libcurl from establishing the connection. Unfortunately Linux does allow regular users(it allows root to do so) to bind to an interface by name, instead it allows the user to bind to an ip address. The ifidx2ip() and netlink_* methods described bellow, describe how to translate an interface name to an ip address.

Prior to binding the socket if this is an IPv6 socket this method sets the socket option IPV6_V6ONLY. This prevents linux from providing a socket compatibility mode that would allow a IPv6 sockets to connect to an IPv4 address, this is necessary to prevent libcurl from bypassing our binding.

The preferred method to achieve the described behavior would be to set the BINDTODEVICE option in the communication socket, however this is only available for the root user(or a user with the CAP_NET_RAW capability). This implementation guarantees that all traffic sent and received through the socket goes through the appropriate interface for the root user, for any other user the intended behavior will not occur if the interface has a global IPv6 address. While this is enough for this implementation, future implementations of this concept should take into

consideration the ongoing work in the Linux kernel regarding *capabilities*..

ifidx2ip(ifidx, family, sock_addr) - This method was created to map an interface name to an IP address in that interface. While Linux already provides methods to do this, they only retrieve IPv4 addresses associated with an interface. However the Linux kernel provides a communication mechanism called netlink, that one can use to query the kernel for information. This method takes an interface index(it can easily be obtained from the name of the interface) as an argument, the family argument specifies the kind of address family to be fetched(v4 or v6) and the sock_addr argument provides storage space to hold the result.

This method will only retrieve the first address associated with the interface, if the interface has multiple addresses the remaining will be ignored. For IPv6 the kernel will return the global IPv6 addresses prior to the link local ones so if a global IPv6 address exists for the interface, this will be returned.

In order to ease the communication with the kernel using netlink three additional methods were created to handle netlink communication:

- netlink_sendRequest()
- netlink_parseMessage()
- netlink_readResponse()

netlink_sendRequest() - This method is provided as a helper for netlink communication. The method creates and sends a request through a given netlink socket requesting a mapping of IP addresses to network interfaces. The response will be stored in a passed buffer.

netlink_readResponse() - This method is provided as a helper for netlink communication. The method blocks while listening on the given socket for a response to request made by the netlink_sendRequest() method. The resulting message is stored in a given buffer.

netlink_parseMessage() - This method is provided as a helper for netlink communication. The method parses the message stored in the given buffer by the netlink_readResponse() method. The message will be searched for mappings of IP addresses for a specific address family to the interface with a given index. The first occurrence of an IP address for the desired interface will be stored in a passed buffer.

4.3.5 Identity Selector gtk widget

Files: WebKit/WebKit/gtk/webkit/identityselector.cpp
WebKit/WebKit/gtk/webkit/identityselector.h
WebKit/WebKit/gtk/webkit/webkit.h
WebKit/WebKit/gtk/webkit/webkitdefines.h

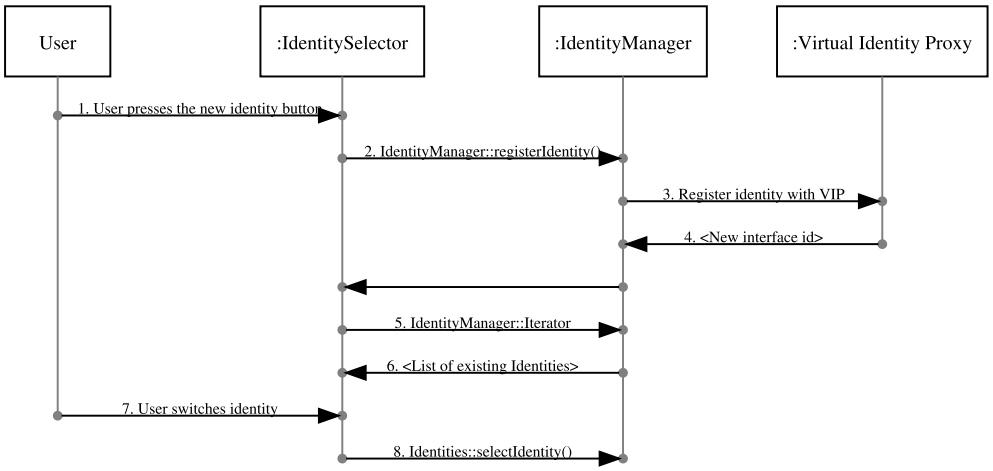

Figure 4.9: Interaction between the Identity Selector and the IdentityManager

Besides the changes in WebKit's internal classes, a gtk widget that can be inserted in gtk graphical interfaces was created to allow a user to create new identities and select existing identities. This widget merely exposes methods from the IdentityManager class to the user.

The user can create and switch identities by selecting the desired identity from the combobox, or create new identities by pressing the appropriate button. After creating a new identity, this identity will be automatically selected.

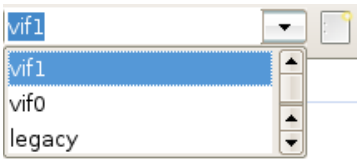

Figure 4.10: Identity Selector widget, listing the legacy identity and two additional identities

4.3.6 WebKit based browser implementations

Files: WebKit/WebKitTools/GtkLauncher/main.c

WebKit itself is not a browser, instead concrete browser implementations use WebKit as an external library, usually building additional graphical interface necessary to each implementation.

In order to introduce these changes to a working browser one must make minor changes to the browser, to display the Identity Selector widget.

Initial tests were made using a demonstration browser that accompanies WebKit, later the implemented changes were tested with a second WebKit based browser called midori[8]. The only changes introduced in both browser implementations were the inclusion of the widget in the interface, other than that the WebKit changes are transparent to the browser implementation.

4.4 Identity Selector for Legacy applications

While the implementation using webkit demonstrates how identity support can be built into applications, enabling identity support in all applications will be a lengthy process. This is required because L7 identifiers are managed by each particular implementation, each requiring specific modifications. This process will certainly not be instantaneous, many applications might not even support this concept at all.

With this in mind, an Identity Selector was implemented, which allows the user to constrict one chosen application to a specific network interface. Used in conjunction with VIP's virtual interfaces, this would allow the user to select L2 and L3 identifiers from a specific identity prior to launching an application. This of course is not a sufficient replacement for full support in the application, but it might be enough, for example for an application that is only used for one of the user's identities. This implementation is divided in two parts:

1. Identity Selector interface
2. Preload shared library

4.4.1 Identity Selector

The Identity Selector(4.11) is the graphical interface that allows the user to launch an application. Upon launch it will list all available network interfaces, and prompt the user to choose one. After the user chooses the interface, the Identity Selector sets the environment variable LD_PRELOAD with the location of the preload_socket library and the variable VIP_INTERFACE with the name of the selected interface. This interface was implementing using bash scripting and zenity to create the interface.

4.4.2 Preload shared library

The linux dynamic linker allows the loading of additional libraries at run time, that superseed existing methods. This shared library is meant to be used in such a way, by the Identity Selector. The library itself implements a wrapper around the socket system call, such that each time the

Figure 4.11: Identity Selector, choosing which VIP interface to use to launch the Firefox browser

target application attempts to create a new socket to establish a new connection, the implemented function intercepts(4.12) the call and binds the socket to a specific interface.

The name of the interface to be used is passed to the library by the Identity Selector, through the VIP_INTERFACE environment variable. This library shares some common source code with the browser implementation. Specifically code related to resolving the interface name into a usable address, that can be used with the bind system call.

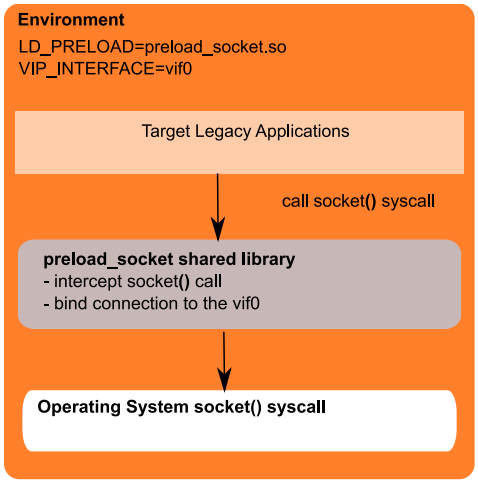

Figure 4.12: Use of LD_PRELOAD to implement legacy support

The implemented shared library reuses much of the code used for binding to the interface in the browser implementation(ResourceHandleManager class), similarly to what happened in the browser implementation these methods are used to bind to a specific interface. Other than that the shared library only overloads the socket() and connect() syscalls. The replacement methods works in a similar fashion to the implementation changes made to the browser implementation.

The method resolves the given interface name to an IP address and binds the socket to that address.

4.4.3 Legacy applications with Profile support

Some applications already support a primitive notion of context separation by identity. The notion of "profiles" is supported by some applications in order to provide support for multiple users using the same application. Application profiles typically provide completely different contexts for different users, including different configuration settings. This separation of user contexts will usually result in completely different L7 identifiers for each user. For example, the Firefox browser supports the concept of user profiles, and each profile has its own cache and cookie storage.

While this is hardly ideal, since the user would have to configure the application for each identity, this could be combined with the devised identity selector for legacy applications in order to provide fully unlinkable identities. This approach has the added inconvenience that, while in the browser implementation presented earlier the browser mapped identities in the VIP to the browser, in this approach there would be no automatic mapping between the VIP interfaces and the browser profiles and the user would have to remember these mappings.

Chapter 5

Implementation Testing and Results

5.1 Test Scenarios

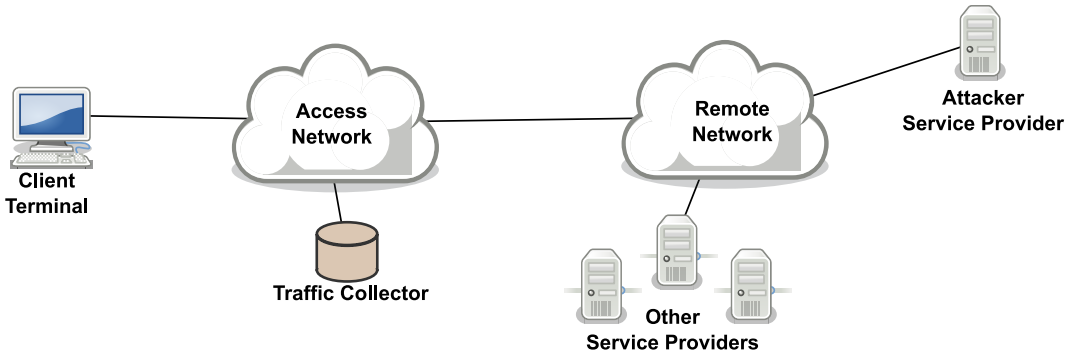

Figure 5.1: Network considered for the conducted tests

The proposed implementations were tested in scenarios with one client terminal in an Access Network, one Attacker Service Provider in a Remote Network as well additional Service Providers. The client terminal was tested using the various proposed implementations seen in chapter 4 as well as regular browser implementations. Traffic going through the Access Network was captured using *tcpdump*. *Tcpdump*'s default behavior is to capture only the packet headers, in order to specify that the entire contents of all packets was to be captured appropriate parameters had to be passed to *tcpdump*(fig. 5.2). The Attacker Service Provider employed the implementation described in section 3.2.4.

```
tcpdump -i eth0 -s 0 -w output.capture
```

Figure 5.2: Command line parameters passed to *tcpdump* in order to capture the full packet length

For the purposes of this scenario the user making use of the client terminal assumes two distinct identities, I_0 and I_1. Under each identity, the user visits one set of web sites, the set S_0 for the identity I_0 and the set S_1 for identity I_1(5.1).

set S_0 visited by identity I_0	set S_1 visited by identity I_1
http://www.sapo.pt/	http://www.ua.pt/
http://www.google.com/	http://nytimes.com/
http://www.myspace.com/	http://www.usatoday.com/
http://digg.com/	http://www.washingtonpost.com/
http://wordpress.org/	http://www.cnn.com/
http://www.imdb.com/	
http://www.ebay.com/	

Table 5.1: Set of sites visited under each identity

5.1.1 Scenario 1 - No support/Regular Browser

The client terminal does not make use of any of the proposed implementations. Tests were made using Firefox browser(version 3.0) , since there is no concept of identity within the browser for this scenario the user will simply visit both sets of web sites.

5.1.1.1 Test workflow

1. Open the browser
2. Visit set S_0
3. Visit the Attacker Service Provider
4. Close the browser
5. Visit set S_1
6. Visit the attacker Service Provider
7. Close the browser

5.1.2 Scenario 2 - Legacy Identity Selector and Regular Browser

In this scenario the client terminal will have an installation of the VIP daemon, as well as the Legacy Identity Selector. The VIP daemon will provide two virtual interfaces, one for identity I_0 and one for identity I_1, the user is able to launch a regular browser under different VIP identities and under each identity he visits the appropriate set of sites.

5.1.2.1 Test workflow

The following workflow assumes that VIP already has two virtual interfaces in place, *vif0* and *vif1*.

1. Open the browser using the Legacy Identity Selector and select the *vif0* interface
2. Visit set S_0
3. Visit the Attacker Service Provider
4. Close the browser
5. Open the browser using the Legacy Identity Selector and select the *vif1* interface
6. Visit set S_1
7. Visit the attacker Service Provider
8. Close the browser

5.1.3 Scenario 3 - Identity Aware Browser

In this scenario VIP is in place and the browser implementation proposed in the previous chapter is used. Virtual identities(and VIP interfaces) are created from the browser Identity Selector.

5.1.3.1 Test workflow

1. Open the Identity Aware browser implementation
2. Using the Identity Selector create a new identity(by default it will be *vif0*)
3. Visit set S_0
4. Visit the Attacker Service Provider
5. Using the Identity Selector create a new identity(by default it will be *vif1*)
6. Visit set S_1
7. Visit the Attacker Service Provider
8. Close the browser

5.1.4 Scenario 4 - Legacy Identity Selector and Regular Browser with Profiles support

As in the second scenario the Legacy Identity Selector was used, but a browser with profiles support was used instead. This scenario is an attempt to have the same benefits advertised by scenario 3 without using the developed browser implementation. For this scenario the firefox browser was used, since it supports user profiles. Selection of profiles in firefox is made prior to launching the browser by selecting a previously configured profile, for this scenario two profiles will be configured previously.

5.1.4.1 Test workflow

The following workflow assumes that VIP already has two virtual interfaces in place, *vif0* and *vif1*. Additionally the browser must have two different profiles configured P_0 and P_1, to be used for identity I_0 and I_1 respectively.

1. Open the browser using the Legacy Identity Selector and select the *vif0* interface
2. In the browser profile selector select profile P_0
3. Visit set S_0
4. Visit the Attacker Service Provider
5. Close the browser
6. Open the browser using the Legacy Identity Selector and select the *vif1* interface
7. In the browser profile selector select profile P_1
8. Visit set S_1
9. Visit the Attacker Service Provider
10. Close the browser

5.2 Test Results

Identifiers presented in these results like IP addresses, MAC addresses and cookie identifiers may vary between scenarios since IP addresses are attributed dynamically and MAC addresses for the virtual interfaces are generated on a random basis. They are however consistent within the result set for each scenario. Between each test run, all browser caches were cleaned and the databases attached to the attack implementation were properly flushed.

5.2.1 Scenario 1 - No support/Regular Browser

5.2.1.1 Service Provider

In tests for this scenario the service provider successfully stored a cookie in the browser with an unique identifier. The Service Provider linked this cookie identifier to the following cache contents:

http://www.sapo.pt/
http://www.google.com/
http://www.ua.pt/
http://www.imdb.com/
http://www.myspace.com/
http://www.usatoday.com/
http://www.washingtonpost.com/
http://digg.com/
http://www.cnn.com/
http://wordpress.org/
http://www.ebay.com/
http://nytimes.com/

The presented set contains all sites in S_0 and all sites in S_1. In practice the Service Provider only "sees" one identity associated with the both web sites sets, S_0 and S_1. This was the expected result, since the purpose of the Service Provider implementation was exactly to correlate information across multiple identities.

5.2.1.2 Access Network

The client terminal always uses the same L2 address, thus all traffic is quickly identified as belonging to the same terminal. Results acquired at the access network provided no additional insight than the ones acquired by the Service Provider(except for the L2 identifiers).

5.2.2 Scenario 2 - Legacy Identity Selector and Regular Browser

5.2.2.1 Service Provider

When employing the Legacy Identity Selector, the user has in fact two different IP addresses. However the primary method for identity correlation in the Service Provider implementation is through cookies. The Service Provider successfully correlates the identities through an identifier placed inside a cookie. As in the previous scenario the Service Provider identifies only one user and successfully associates the following browser cache content with the user:

http://www.sapo.pt/
http://www.google.com/
http://www.ua.pt/
http://www.imdb.com/
http://www.myspace.com/
http://www.usatoday.com/
http://www.washingtonpost.com/
http://digg.com/
http://www.cnn.com/
http://wordpress.org/
http://www.ebay.com/
http://nytimes.com/

From the Service Provider perspective, this is almost analogous to the previous scenario, the only different being that two IP addresses were recorded in the database.

5.2.2.2 Access Network

Traffic captured in the Access Network reveals the presence of two different MAC addresses in the network, each belonging to one of the virtual interfaces in the client terminal(*vif0* and *vif1*). Associated to each MAC address is a different IP address, as if two different client terminals existed in the network which was in fact the intended effect for this scenario.

5.2.3 Scenario 3 - Identity Aware Browser

5.2.3.1 Service Provider

In this scenario the Service Provider identified two distinct users, each with different IP addresses and cookie identifiers. The Service Provider implementation also determined that the following sites where in the first user's browser cache:

http://www.sapo.pt/
http://www.google.com/
http://www.myspace.com/
http://digg.com/
http://wordpress.org/
http://www.imdb.com/
http://www.ebay.com/

Which corresponds exactly to the site set S_0 defined previously. The second user identified by the Service Provider was found to have the following set of web sites in the browser's cache:

http://www.ua.pt/
http://www.usatoday.com/
http://www.washingtonpost.com/
http://www.cnn.com/
http://nytimes.com/

Which matches the set S_1. From the Service Provider's perspective it actually appears that two different users exist which was the desirable result for this scenario.

5.2.3.2 Access Network

As in the previous scenario, traffic captured in the access network suggests that two terminals exist in the network. Figure 5.3 depicts the associations between IP and MAC addresses in the access network, as determined by the developed attack implementation. Items marked in gray are the relevant addresses for this scenario, items in white are other terminals in the network.

As intended two distinct MAC and IP addresses are seen in the network. In figure 5.3 the MAC address *00:FF:7D:D2:32:A1* is associated with the identity I_1(interface vif1 in the client terminal) and the MAC address *00:FF:FD:F4:CE:2C* is associated with the identity I_0(interface vif0 in the client terminal).

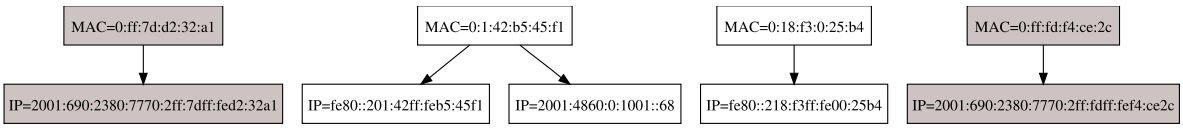

Figure 5.3: Association between IP and MAC addresses in the access network for a test run of the third scenario. Items in gray are the addresses associated with each user identity.

5.2.4 Scenario 4 - Legacy Identity Selector and Regular Browser with Profiles support

Results regarding the fourth scenario were identical to the ones seen for the third scenario. As with the previous configuration, traffic captured at the Access Network shows that traffic under each identity employs distinct MAC and IP addresses. The Service Provider associated implementation identified two identities and associated the set S_0 with the identity I_0 and the set S_1 to the identity I_1.

5.2.5 Summary

Table 5.2 shows the overall results for the various scenarios. As was expected, the configuration for scenarios 1 and 2 make the user vulnerable to identity correlation attacks while the configuration

seen in scenarios 3 and 4 do not. Scenario 4 also has some disadvantages when compared to the third scenario, since it forces the user to restart the browser each time he wants to switch identities and forces the user to remember the association between the virtual identities as created by the VIP and the browser profiles.

	L7 based collusion	L3 based collusion	L2 based collusion
Scenario 4: Legacy Identity Selector + Browser with Profiles support	Not across different identities	Not across different identities	Not across different identities
Scenario 3: Identity Aware Browser	Not across different identities	Not across different identities	Not across different identities
Scenario 2: Legacy Identity Selector + Regular browser	Vulnerable	Not across different identities	Not across different identities
Scenario 1: Regular browser	Vulnerable	Vulnerable	Vulnerable

Table 5.2: Overall test results

Chapter 6

Conclusions

The pursue for more suitable identity models for the Internet, has resulted in some very capable proposals based on the federated identity model. All the presented frameworks have matured to the point where Service Providers are now considering wide adoption and seem intent on converging torwards a model that promotes interoperability between them. These frameworks have already succeeded in introducing the federated identity model to the consumers and users seem to be eager to benefit from some of the features they provide, like Single Sign On and omnidirectional pseudonyms. This model does enable support for more robust authentication for the user and more control over the dissemination of user information. However while the reviewed frameworks do recognize that identifiers used for each identity must be unrelated for the user to have unlinkable identities, they only address this requirement within the identity framework itself and neglect that the underlying layers do not support any kind of notion of identity. Since they fail to extend these privacy guarantees to the lower layers, the user will never truly benefit from the added privacy features they advertise.

As it was seen for the user to have guarantees in terms of unlinkable identities, a cross layer approach that takes into consideration identifiers used in all layers must be considered. System support must be available to provide L2 and L3 identifiers, similarly to what the VIP implementation does and individual applications must also provide support for unlinkable L7 identifiers. L7 protocol Identifiers and caching mechanisms vary considerably between protocols and application implementations, making the task of implementing support hardly trivial. This work outlines the necessary requirements for identity privacy, and proposes a demonstration implementation for a browser that enables identity privacy by integrating with an existing implementation. A similar approach could be considered for different applications.

6.1 Further Considerations

The presented browser implementation is an example of how to support identity privacy within a particular application. However nowadays it is common for applications in the user's desktop to communicate among themselves. Browsers for example, usually invoke external applications to handle features like video streaming or support for applets, this poses the problem of making different applications cooperate using the same user identity. This would require that applications could signal to one another, the appropriate identity to be used. This work however would only

be addressable, if multiple different applications already supported multiple identities for the user.

The work presented here, implements unlinkable identities across all layers of the OSI stack. However technologies like OpenID and SAML lie above the Application Layer and it would be desireable that integration of these technologies with the work presented here would be possible. Ideally in such a way that whenever the user selected an identity from one of these frameworks, the privacy requirements in the underlying layers as described in this work were immediately enforced.

While the all the software implementations used in this work works in the *userspace*, there is some ongoing work in the Linux kernel regarding support for multiple network stacks usually named "network namespaces". While network namespaces in the Linux kernel have been implemented mainly with virtualisation and security concerns in mind, support for distinct network stacks directly in the operating system would be a welcome addition for future implementations that addressed identity privacy.

Bibliography

[1] http://identifight.org/.

[2] Bandit project. http://www.bandit-project.org/.

[3] Bug 147777 – :visited support allows queries into global history. https://bugzilla.mozilla.org/show_bug.cgi?id=147777.

[4] Google Analytics. http://www.google.com/analytics/.

[5] The gtk+ project. http://www.gtk.org/.

[6] Higgins project. http://www.eclipse.org/higgins/.

[7] libcurl - the multiprotocol file transfer library. http://curl.haxx.se/libcurl/.

[8] "midori - web browser". http://software.twotoasts.de/?page=midori.

[9] "SQLite". http://www.sqlite.org/.

[10] "TCPDUMP". http://www.tcpdump.org.

[11] Webkit. http://webkit.org/.

[12] *Identity federation for voip-based services* (New York, NY, USA, 2007), ACM.

[13] Alvestrand, H. Tags for the Identification of Languages. RFC 3066 (Best Current Practice), Jan. 2001. Obsoleted by RFCs 4646, 4647.

[14] Berners-Lee, T., Fielding, R., and Masinter, L. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), Jan. 2005.

[15] Cameron, K. *The Laws of Identity*. Microsoft Corporation, May 2005.

[16] Cantor, S. Shibboleth Architecture: Protocols and Profiles. http://shibboleth.internet2.edu/docs/internet2-mace-shibboleth-arch-protocols-200509.pdf, September 2005.

[17] Cantor, S., Hirsch, F., Kemp, J., Philpott, R., and Maler, E. Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. http://docs.oasis-open.org/security/saml/v2.0/, March 2005.

[18] Cantor, S., Kemp, J., Philpott, R., and Maler, E. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. http://docs.oasis-open.org/security/saml/v2.0/, March 2005.

[19] Chung, W., and Paynter, J. Privacy Issues on the Internet. In *HICSS* (2002), p. 193.

[20] Clarke., R. Identified, Anonymous and Pseudonymous Transactions: The Spectrum of Choice. In *User Identification & Privacy Protection Conference* (1999).

[21] Council, E. Directive 2002/58/ec of the european parliament and of the council of 12 july 2002 concerning the processing of personal data and the protection of privacy in the electronic communications secto (directive on privacy and electronic communications). *Official Journal of the European Communities* (2002). L201/37 - L201/47.

[22] Curbera, F., Schlimmer, J., Ballinger, K., Box, D., Graham, S., Liu, C. K., Lovering, B., Nadalin, A., Nottingham, M., Orchard, D., von Riegen, C., Shewchuk, J., Truty, G., and Weerawarana, S. Web Services Metadata Exchange (WS-MetadataExchange), February 2004.

[23] Eastlake, D., and Reagle, J. XML Encryption Syntax and Processing. http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/, December 2002.

[24] Eastlake 3rd, D., Reagle, J., and Solo, D. (Extensible Markup Language) XML-Signature Syntax and Processing. RFC 3275 (Draft Standard), Mar. 2002.

[25] Felten, E. W., and Schneider, M. A. Timing attacks on Web privacy. In *ACM Conference on Computer and Communications Security* (2000), pp. 25–32.

[26] for Economic Cooperation, O., and (OECD), D. The guidelines on the protection of privacy and transborder flows of personal data. *OECD Council Recomendation* (1980).

[27] Girao, J., Alfredo Matos, Sargento, S., and Aguiar, R. L. Preserving Privacy in Mobile Environments With Virtual Network Stacks. In *50th Annual IEEE Global Telecommunications Conference* (Washington, DC, USA, November 2007), GLOBECOM 2007.

[28] Gomi, H. Session Management for Cross-protocol Identity Federation. *Web Services, 2007. ICWS 2007. IEEE International Conference on* (9-13 July 2007), 1150–1151.

[29] Hardt, D., and Bufu, J. Openid information cards 1.0 - draft 01. https://openidcards.sxip.com/spec/openid-infocards.html, August 2007.

[30] Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., and Maler, E. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. http://docs.oasis-open.org/security/saml/v2.0/, March 2005.

[31] ITU-T Correspondence Group. Report on the Definition of the Term "Identity", 2008.

[32] Jackson, C., Bortz, A., Boneh, D., and Mitchell, J. C. Protecting browser state from web privacy attacks. In *WWW* (2006), L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, Eds., ACM, pp. 737–744.

[33] Krishnamurthi, G., and Chan, T.-K. Using the liberty alliance architecture to secure ip-level handovers. In *COMSWARE* (2006), IEEE.

[34] Krishnamurthy, B., Malandrino, D., and Wills, C. E. Measuring privacy loss and the impact of privacy protection in web browsing. In *SOUPS '07: Proceedings of the 3rd symposium on Usable privacy and security* (New York, NY, USA, 2007), ACM, pp. 52–63.

[35] Kristol, D., and Montulli, L. HTTP state management mechanism. RFC 2109, Internet Engineering Task Force, Feb. 1997.

[36] Maler, E., and Reed, D. The Venn of Identity: Options and Issues in Federated Identity Management. *Security & Privacy, IEEE 6*, 2 (March-April 2008), 16–23.

[37] Matos, A., Sargento, S., and Aguiar, R. Embedding identity in mobile environments. In *MobiArch '07: Proceedings of first ACM/IEEE international workshop on Mobility in the evolving internet architecture* (New York, NY, USA, 2007), ACM, pp. 1–8.

[38] Miller, J. Yadis Specification 1.0. http://yadis.org/papers/yadis-v1.0.pdf, March 2006.

[39] Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and Granqvist, H. WS-SecurityPolicy 1.2. `http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.pdf`, July 2007. OASIS Web Services Secure Exchange Technical Committee.

[40] Nadalin, A., Goodner, M., Gudgin, M., Barbir, A., and Granqvist, H. WS-Trust 1.3. `http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.pdf`, March 2007. OASIS Web Service Secure Exchange Technical Committee:.

[41] OpenID Authentication 2.0. `http://openid.net/specs/openid-authentication-2_0.html`, December 2007.

[42] Patel, V., and Juric, R. Internet users and online privacy: a study assessing whether Internet users' privacy is adequately protected. vol. 1, ITI, pp. 193 – 200.

[43] Pfitzmann, A., and Hansen, M. Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology (version v0.31). `http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.31.pdf`, February 2008.

[44] Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., and Scavo, T. Security assertion markup language (SAML) v2.0 technical overview working draft 21 february 2007. `http://www.oasis-open.org/committees/download.php/22553/sstc-saml-tech-overview-2%200-draft-13.pdf`, February 2007.

[45] Recordon, D., and Fitzpatrick, B. Openid authentication 1.1. `http://openid.net/specs/openid-authentication-1_1.html`, May 2006.

[46] Reed, D., and McAlpin, D. Extensible resource identifier (XRI) syntax v2.0. Organization for the Advancement of Structured Information Standards (OASIS), Committee Specification, Nov. 2005.

[47] Sicker, D., Kulkarni, A., Chavali, A., and Fajandar, M. A federated model for secure Web-based videoconferencing. *Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on* (28-30 April 2003), 396–400.

[48] Springer. *Virtual Identity Framework for Telecom Infrastructures* (Netherlands, February 2008). ISSN 0929-6212.

[49] Wason, T. Liberty Alliance Project. Liberty ID-FF Architecture Overview (Version 1.2-errata-v1.0). `http://www.projectliberty.org/liberty/resource_center/specifications`, 2005. Accessed at February 2008.

[50] Weitzner, D. J. Whose Name Is It, Anyway? Decentralized Identity Systems on the Web. *IEEE Internet Computing 11*, 4 (2007), 72–76.